Development and implementation of a browser extension for phishing attack prevention

By

Aruzhan Rakhymzhanova, Moldir Yerlan, Zhuldyz Myrzagalieva

Department of Intelligent Systems and Cybersecurity
Astana IT University

6B06301 — Cybersecurity
Supervisor: Mukusheva Saniya

June 2025
Astana

Abstract

This work presents a comprehensive solution for identifying and preventing phishing attacks through real-time browser-based detection mechanisms. Through extensive literature analysis and empirical development, the study addresses the persistent challenge of web-based social engineering attacks, where current statistics indicate that phishing incidents have exceeded 1.2 million documented cases, with traditional security measures failing to provide adequate protection against evolving deceptive techniques.

The research begins by examining the evolution of phishing attack vectors from email-based schemes to sophisticated multi-platform campaigns, establishing a theoretical foundation through analysis of existing detection methodologies and their limitations in contemporary threat landscapes. The study identifies critical gaps in current defensive approaches, particularly the vulnerability windows created by static rule-based systems and the performance constraints of existing machine learning implementations in real-time environments.

Additionally, the research includes a comprehensive evaluation of browser extension architectures and their effectiveness in client-side threat mitigation, analyzing key factors that influence detection accuracy and user adoption rates. The study introduces an innovative methodology combining supervised machine learning with threat intelligence integration and DOM-based content analysis.

The practical implementation involves developing a Chrome browser extension utilizing Random Forest classification trained on 6.5 million URL samples, achieving 96.2% precision with 93.7% recall performance metrics. The system integrates Google Safe Browsing and VirusTotal APIs for external threat validation while implementing real-time webpage analysis to detect cloned authentication interfaces and deceptive visual elements.

The study concludes with comprehensive evaluation results demonstrating ROC-AUC scores of 0.983 and false positive rates below 0.1%, validated through user experience assessment involving thirty participants. Performance testing confirms the system's effectiveness in detecting zero-day phishing attempts while maintaining minimal computational overhead.

This work contributes to both the theoretical understanding of client-side security mechanisms and provides practical tools for enhancing web browsing security in real-world computing environments. The modular, privacy-preserving architecture offers immediate deployment capabilities while supporting future adaptations to emerging threat vectors.

Definitions

The following terms are used in this work:

- **Phishing** – A cyberattack technique that deceives users into revealing confidential information by impersonating trusted entities through fake websites, emails, or messages.
- **Machine Learning** – A method in artificial intelligence that enables systems to learn from data and identify patterns for making predictions.
- **Browser Extension** – A lightweight software add-on that enhances browser functionality, used here to detect phishing attempts during real-time browsing.
- **Document Object Model** – A hierarchical model representing the structure of web pages, used for inspecting suspicious elements such as fake login forms.
- **Application Programming Interface** – A communication protocol that enables data exchange between software components; used to integrate threat intelligence.
- **Random Forest** – An ensemble-based machine learning algorithm that builds multiple decision trees for classification, chosen here for its robustness and high accuracy.
- **False Positive** – An instance where a legitimate website is incorrectly classified as phishing, leading to unnecessary user warnings.
- **Supervised Learning** – A training approach where the model learns to classify inputs based on pre-labeled examples.
- **Threat Intelligence** – Data collected from external sources regarding known threats, which is used to validate suspicious URLs.
- **URL (Uniform Resource Locator)** – The address of a web page; structural and lexical characteristics of URLs are used as phishing indicators.

Designations and Abbreviations

The following designations and abbreviations are used in this work:

- **ML**: Machine Learning
- **DOM**: Document Object Model
- **API**: Application Programming Interface
- **UI**: User Interface
- **ROC-AUC**: Receiver Operating Characteristic – Area Under Curve
- **URL**: Uniform Resource Locator
- **GSB**: Google Safe Browsing
- **VT**: VirusTotal
- **CSP**: Content Security Policy
- **JS**: JavaScript

List of Tables

Table **Page**

List of Figures

Table of Contents

Chapter 1

INTRODUCTION

1.1 Relevance

The contemporary digital landscape faces unprecedented challenges from sophisticated cyber-attack methodologies, with deceptive web-based schemes representing a critical vector for credential harvesting and organizational security breaches. Current threat intelligence demonstrates an alarming escalation in malicious campaign frequency, with documented incidents exceeding 1.2 million unique attack vectors throughout 2023, marking a substantial increase from previous reporting periods.

Financial institutions and their associated digital ecosystems continue to experience disproportionate targeting through these attack mechanisms, resulting in cumulative global economic losses surpassing 50 billion US dollars annually. This economic impact extends beyond direct monetary theft to encompass reputation damage, regulatory penalties, operational disruption, and remediation costs that collectively threaten organizational viability across multiple industry sectors. Traditional defensive architectures demonstrate significant limitations when confronting evolved attack methodologies that employ dynamic domain generation, sophisticated visual mimicry, and contextually adaptive deception techniques. These advanced approaches systematically circumvent conventional rule-based detection systems, creating substantial vulnerability windows during critical user decision-making moments.

Web browsers function as primary attack surfaces where users encounter malicious content, representing the most frequent interaction point between potential victims and deceptive schemes. Despite considerable investments in server-side security infras-

tructure and comprehensive user education programs, attackers successfully exploit the psychological and technical vulnerabilities inherent in real-time browsing scenarios.

The critical nature of immediate intervention during user-threat interactions necessitates advanced client-side protective mechanisms capable of real-time threat assessment and user notification. This requirement establishes browser-based security extensions as essential components within comprehensive cybersecurity architectures, offering unique capabilities for contextual threat analysis and immediate protective intervention.

1.2 Objectives and Goals

Primary Goal

This research initiative aims to develop and implement a high-performance browser extension architecture capable of detecting and neutralizing phishing attacks through integrated multi-layered analytical methodologies. The fundamental objective encompasses achieving superior detection accuracy exceeding 95%, maintaining false positive rates below 0.1%, and ensuring response latency remains under 250 milliseconds to preserve optimal user experience while delivering effective security protection.

Specific Research Objectives

- **Architectural Development Objective**: Design and implement a modular, scalable extension framework supporting concurrent detection modules without compromising browser performance or system stability. This architecture must accommodate future enhancements and threat evolution while maintaining backward compatibility across Chromium-based browser platforms.
- **Machine Learning Implementation Objective**: Develop and deploy supervised learning algorithms utilizing Random Forest classification methodologies for automated URL threat assessment. This implementation will focus on identifying suspicious lexical patterns, domain characteristics, and structural anomalies indicative of malicious intent.
- **External Intelligence Integration Objective**: Establish seamless connectivity with authoritative threat intelligence services including Google Safe Browsing API and VirusTotal platforms. This integration enables real-time cross-referencing

of suspicious resources against comprehensive malicious domain databases and reputation scoring systems.

- **Content Analysis Implementation Objective**: Implement sophisticated Document Object Model analysis capabilities for identifying deceptive webpage elements including cloned authentication interfaces, concealed input mechanisms, and visually misleading components commonly employed in phishing scenarios.
- **User Experience Optimization Objective**: Design and implement intuitive user interface components delivering contextually appropriate, non-intrusive security notifications based on dynamic risk assessment algorithms. This interface must effectively communicate threat levels without disrupting normal browsing workflows.
- **Performance Validation Objective**: Conduct comprehensive testing protocols utilizing extensive datasets encompassing both legitimate and malicious web resources to validate system accuracy, efficiency, and reliability under diverse operational conditions.

1.3 Object and Subject of Research

Research Object

The primary research object encompasses phishing attack methodologies as they manifest within contemporary web browsing environments, with particular emphasis on technical implementation strategies employed by threat actors and the corresponding defensive detection opportunities available within browser-based security frameworks.

This research object includes analysis of attack vector evolution, from rudimentary email-based deception campaigns to sophisticated multi-platform schemes incorporating advanced social engineering techniques, visual mimicry, and contextual awareness. The investigation encompasses technical mechanisms underlying successful attacks, including domain manipulation strategies, SSL certificate exploitation, redirection techniques, and psychological manipulation methodologies employed to bypass user skepticism.

Research Subject

The research subject focuses specifically on the development and optimization of defensive browser-based security tools, with primary emphasis on architectural design principles

and algorithmic implementation strategies for real-time phishing detection systems.

This subject encompasses the technical requirements for effective client-side security implementations, including performance optimization techniques, user experience design principles, and integration methodologies for external threat intelligence services. The research subject extends to evaluation metrics for measuring detection effectiveness, false positive minimization strategies, and scalability considerations for enterprise deployment scenarios.

1.4 Methodological Foundations of the Study

Quantitative and Qualitative Analysis Framework

Qualitative and Quantitative Analysis This research integrates both qualitative and quantitative methodologies to ensure a comprehensive evaluation of phishing detection mechanisms. On the quantitative side, statistical techniques are used to measure system performance across metrics such as accuracy, precision, recall, and F1-score. Cross-validation, receiver operating characteristic (ROC) analysis, and area under curve (AUC) calculations help validate the robustness of the machine learning models. Additional quantitative metrics include response time, resource utilization, and scalability under varying computational loads. Data from structured Google Forms will also be analyzed to quantify user feedback, system usability, and practical effectiveness.

On the qualitative side, supervised machine learning techniques are employed, leveraging labeled datasets of malicious and legitimate URLs. The methodology involves lexical content analysis, domain reputation checks, certificate validation, and redirection pattern recognition. The Random Forest algorithm, enhanced with ensemble learning principles, is implemented to increase accuracy while reducing overfitting. Training involves hyperparameter tuning, feature importance ranking, and qualitative assessments of model generalization across diverse threat environments.

Supervised Learning Methodology

URL threat assessment utilizes supervised machine learning approaches trained on extensive labeled datasets containing both malicious and legitimate web resources. Feature extraction methodologies focus on lexical content analysis, domain reputation scoring,

certificate validation, and redirection pattern recognition.

The Random Forest algorithm implementation incorporates ensemble learning principles to enhance detection accuracy while minimizing overfitting risks. Model training procedures include hyperparameter optimization, feature importance analysis, and cross-validation protocols to ensure generalization capabilities across diverse threat landscapes.

API Integration Framework

External threat intelligence integration employs RESTful API communication protocols with established security services including Google Safe Browsing and VirusTotal platforms. This methodology enables real-time threat verification through authoritative blacklist databases and reputation scoring systems.

Integration protocols incorporate error handling mechanisms, rate limiting compliance, and fallback procedures to ensure system reliability during external service interruptions. Security considerations include encrypted communication channels and minimal data exposure principles to protect user privacy.

DOM Analysis Techniques

Document Object Model analysis employs systematic webpage structure examination to identify deceptive elements commonly associated with phishing attacks. This methodology includes form analysis, link verification, visual similarity assessment, and suspicious element detection algorithms.

Content analysis techniques incorporate pattern recognition algorithms for identifying cloned login interfaces, hidden form fields, and misleading visual components. The methodology includes dynamic content monitoring to detect real-time modifications indicative of malicious behavior.

1.5 Characteristics of Scientific and Practical Activities

Scientific Contribution Characteristics

This research contributes to cybersecurity knowledge through empirical validation of multi-layered detection approaches in real-world browser environments. The scientific value encompasses advancement of client-side security methodologies, machine learning applications in threat detection, and user experience optimization in security tool design.

The investigation provides quantitative evidence for the effectiveness of hybrid detection systems combining local machine learning analysis with external threat intelligence validation. Research findings contribute to understanding optimal balance points between detection accuracy and system performance in resource-constrained browser environments.

Practical Implementation Activities

Development activities encompass complete browser extension implementation from initial architecture design through final testing and validation phases. Practical work includes coding, testing, debugging, and optimization procedures necessary for production-ready software deployment.

Implementation activities involve extensive compatibility testing across multiple browser versions, operating system environments, and hardware configurations. Performance optimization includes memory usage minimization, processing efficiency enhancement, and user interface responsiveness improvement.

Validation and Testing Activities

Comprehensive testing protocols include functionality verification, performance benchmarking, security assessment, and usability evaluation procedures. Testing activities encompass automated unit testing, integration testing, and system-level validation using realistic threat scenarios.

User experience evaluation involves heuristic analysis, cognitive walkthrough procedures, and usability testing with representative user groups. These activities ensure the

extension delivers effective security protection without compromising browsing experience quality.

Educational and Awareness Components

The research incorporates educational elements designed to enhance user understanding of phishing threats and defensive strategies. Practical activities include development of contextual help systems, threat explanation interfaces, and user guidance mechanisms integrated within the extension architecture.

These educational components serve dual purposes of immediate threat mitigation and long-term user security awareness improvement, contributing to overall cybersecurity posture enhancement beyond technical protection mechanisms.

Chapter 2

LITERATURE REVIEW

2.1 Phishing: Conceptual Foundations and Historical Development

2.1.1 The Concept and Definition of Phishing

Phishing has emerged as one of the most persistent and adaptive threats in the cybersecurity domain. Defined as a type of social engineering attack, phishing involves deceiving users into revealing sensitive information by impersonating trusted entities [29]. Phishing represents one of the most insidious forms of social engineering in cyberspace, aimed at gaining unauthorized access to users' confidential information by masquerading as trusted sources. This type of attack is characterized by deliberately deceiving victims into revealing personal data, such as login credentials, credit card numbers, and other sensitive information [5]. The etymology of the term "phishing" originates from "fishing," reflecting the technique's nature—baiting victims with seemingly legitimate communications to "hook" their personal information.

2.1.2 Historical Evolution of Phishing Attacks

Initially observed in the mid-1990s through rudimentary scams targeting AOL users, phishing has undergone significant transformation over the past decades. What began as simplistic email fraud has evolved into a multifaceted threat landscape comprising highly targeted spear-phishing campaigns, mobile attacks, and social media exploits [37]. The historical trajectory of phishing attacks reveals a remarkable evolution in

both sophistication and scale since its inception in the mid-1990s. Initial manifestations emerged in 1996 when attackers targeted America Online (AOL) users, impersonating AOL staff to solicit account verification details [30]. These rudimentary attacks relied on basic social engineering techniques distributed through instant messaging platforms and email systems. As internet adoption accelerated in the early 2000s, phishing operations expanded beyond AOL, targeting banking and financial institutions. The mid-2000s witnessed a significant paradigm shift as phishing transformed from opportunistic crimes to organized criminal enterprises. By 2005, underground phishing "kits" began circulating, enabling even technically unsophisticated individuals to launch convincing attacks [36].

2.1.3 Contemporary Phishing Landscape and Statistics

The period between 2010 and 2020 marked the industrialization of phishing, characterized by increased targeting precision, operational scale, and technical complexity. Recent statistics from the Anti-Phishing Working Group (APWG) indicate that phishing incidents have increased by over 300% since 2020, with over 1.2 million unique phishing campaigns observed in the last quarter of 2024 alone [10]. The COVID-19 pandemic served as a catalyst for this spike, with cybercriminals exploiting widespread anxiety and the rapid transition to remote work environments. Contemporary phishing has evolved far beyond its rudimentary origins. Modern attacks employ polymorphic domains, dynamic page content, and evasive techniques designed to circumvent traditional security controls. The sophistication of these operations is evident in their ability to create convincing visual replicas of legitimate websites, often indistinguishable to the untrained eye [61]. Furthermore, adversaries increasingly leverage artificial intelligence and machine learning to automate attack personalization and enhance deception efficacy [44].

2.2 Taxonomy of Modern Phishing Attack Vectors

2.2.1 Email-Based Phishing Techniques

Email remains the predominant vector for phishing attacks, accounting for approximately 54% of all reported incidents [86]. This prevalence stems from email's universal adoption, low operational costs for attackers, and the ability to conceal deceptive elements. Email-based phishing remains the most common vector, representing more than half of all known incidents [85]. These attacks use a range of techniques, including brand impersonation, business email compromise, and malicious attachments or embedded scripts that evade

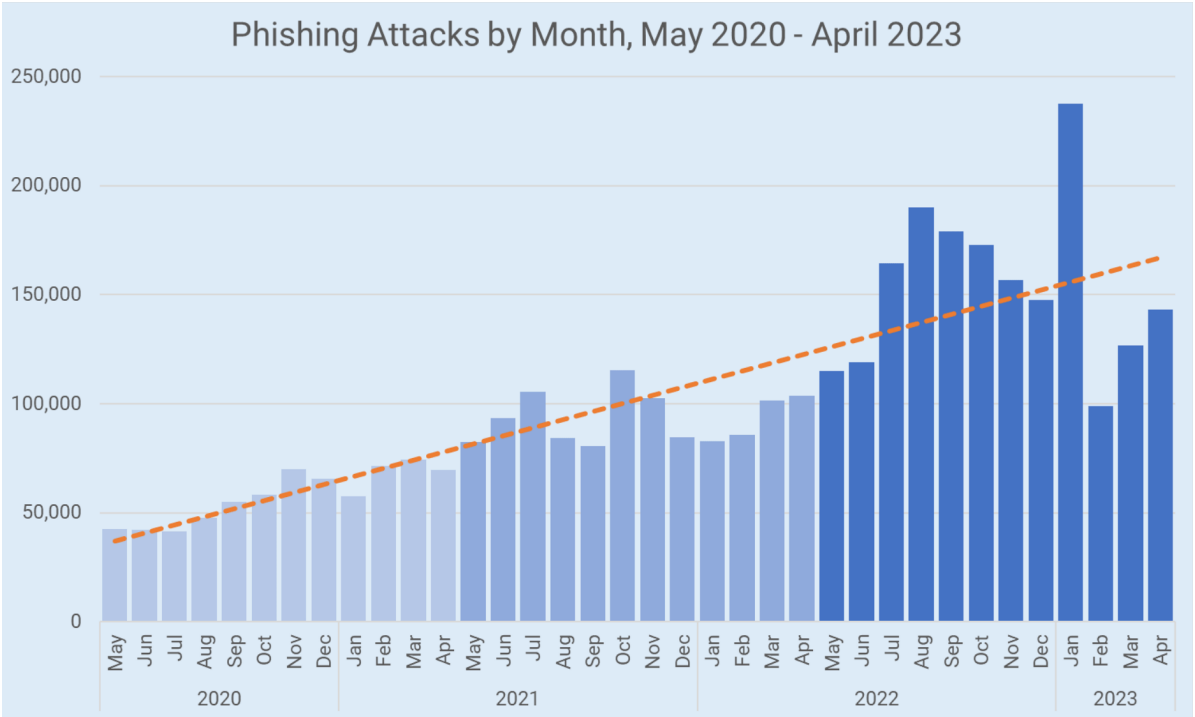


Figure 2.1: Phishing Attacks by month, May 2020 - April 2023

detection [3]. Contemporary email phishing employs several sophisticated techniques:

Business Email Compromise (BEC): This involves the impersonation of executives or trusted business partners to authorize fraudulent transactions or data transfers. BEC attacks have cost organizations worldwide over $43 billion between 2016 and 2024, according to FBI statistics [24].

Spear Phishing: Unlike generic campaigns, spear phishing targets specific individuals or organizations with customized communications containing personalized details harvested from social media or data breaches. Research by Symantec [81] indicates that spear phishing attacks have a success rate nearly 40 times higher than generic phishing attempts.

Whaling: A specialized form of spear phishing directed at high-value targets such as C-suite executives, government officials, or individuals with exceptional access privileges. These attacks feature meticulous research and sophisticated social engineering [8].

Clone Phishing: This technique involves duplicating legitimate communications previously received by targets but replacing original attachments or embedded links with malicious alternatives. The familiarity of the communication format significantly increases success rates [70].

2.2.2 Web-Based Phishing Strategies

Meanwhile, web-based phishing has grown in prominence through the use of visually deceptive websites, look-alike domains, rogue SSL certificates, and cloaking techniques that mask malicious content from automated scanning tools [31]. Web-based phishing represents the second most prevalent attack vector, employing various techniques to deceive users interacting with seemingly legitimate websites:

Lookalike Domains: Attackers register domains visually similar to legitimate websites (e.g., g00gle.com" instead of google.com") exploiting users' tendency to overlook subtle URL discrepancies. Research by CheckPoint [18] documented over 60,000 newly registered lookalike domains targeting major financial institutions in 2023 alone.

Rogue SSL Certificates: The widespread adoption of HTTPS has led attackers to obtain legitimate SSL certificates for fraudulent domains, presenting the security padlock icon that many users associate with trustworthiness. Studies show approximately 84% of phishing websites now deploy SSL certificates [79].

Cross-Site Scripting (XSS): This technique injects malicious scripts into trusted websites, enabling the theft of session cookies or redirection to phishing pages. XSS vulnerabilities remain prevalent, with OWASP consistently ranking them among the top web application security risks [64].

Cloaking Techniques: Advanced phishing sites implement cloaking methods that deliver benign content to security scanners while serving malicious content to actual users. This approach significantly impedes automated detection systems [89].

2.2.3 Mobile-Specific Phishing Vectors

The widespread adoption of mobile devices has opened additional channels for phishing. SMS-based attacks (smishing), QR code phishing, malicious applications, and voice phishing (vishing) exploit mobile interfaces and limited user attention spans [21]. The ubiquity of smartphones has created fertile ground for mobile-specific phishing attacks:

SMS Phishing (Smishing): These attacks utilize text messages containing malicious links or urgent requests. The limited screen size of mobile devices often obscures suspicious URL elements, while the immediate nature of SMS creates a false sense of urgency. IBM Security reported a 328% increase in smishing attempts between 2021 and 2024 [32].

Application-Based Phishing: Malicious applications masquerading as legitimate services infiltrate app stores or are distributed through side-loading. These applications implement convincing interfaces that harvest credentials or financial information. Analysis

by Lookout Security identified over 4,000 malicious financial applications in 2023 alone [48].

QR Code Phishing (Quishing): This emerging vector embeds malicious links in QR codes, exploiting their increasing mainstream adoption. The COVID-19 pandemic accelerated QR code usage, with McAfee documenting a 1,200% increase in QR-based phishing attempts from 2020 to 2024 [51].

Voice Phishing (Vishing): Combining traditional phone scams with digital elements, vishing operations often leverage VoIP services to impersonate legitimate organizations. Advanced vishing employs AI-generated voices to clone authentic representatives, enhancing credibility [70].

2.2.4 Social Media and Emerging Attack Platforms

Simultaneously, social media has become a high-risk vector due to its informal nature and trust-based communication patterns. In recent years, phishing incidents via platforms like Facebook and LinkedIn have significantly increased [39]. The increasing centrality of social media in daily communications has established these platforms as prime phishing vectors:

Account Impersonation: Attackers create convincing replicas of legitimate accounts belonging to trusted entities, friends, or influential figures. These accounts then distribute malicious links or solicit sensitive information. Meta reported disabling over 1.3 billion fake Facebook accounts in 2024, many involved in phishing operations [52].

Malicious Advertisements: Sponsored content featuring phishing links appears in users' feeds, exploiting the perceived legitimacy of the platform's advertising system. Studies indicate users are three times more likely to interact with malicious content when it appears as sponsored material [4].

Direct Message Campaigns: Compromised or fake accounts initiate private conversations containing phishing links, often leveraging existing trust relationships or interest-based targeting [30]. Newer technologies such as augmented reality are also being explored for phishing, with attackers using deceptive visual overlays to mislead users [17]. Several emerging phishing vectors demonstrate the continuous evolution of this threat:

Augmented Reality (AR) Phishing: As AR applications gain mainstream adoption, researchers have identified potential exploitation vectors where malicious overlays can be superimposed on legitimate physical objects, such as QR codes or product packaging [95]. **Machine Learning-Enhanced Phishing**: Adversaries increasingly leverage machine learning algorithms to generate highly convincing phishing content customized to

individual targets. These systems analyze behavioral patterns and communication styles to craft personalized lures [45].

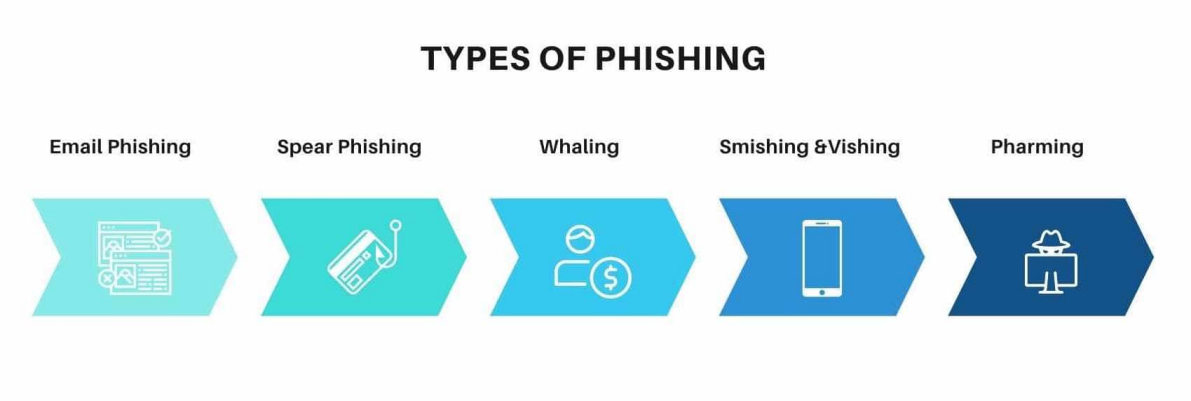


Figure 2.2: Types of Phishing

2.3 Anti-Phishing Defense Mechanisms and Technologies

2.3.1 Network and Server-Side Defense Solutions

To counteract phishing, several defense mechanisms have been introduced. The response to evolving phishing threats encompasses technological solutions, procedural frameworks, and educational initiatives. These solutions operate at infrastructure levels, attempting to identify and neutralize phishing attempts before they reach end-users:

Email Filtering Systems: Modern email security gateways employ multiple detection techniques, including sender reputation analysis, content inspection, and behavioral anomaly detection. Advanced systems utilize machine learning algorithms to identify subtle indicators of phishing attempts, achieving detection rates of 85-95% for known attack patterns [69].

URL Blacklisting Services: Blacklist-based systems provide rapid detection for known phishing domains but suffer from latency in identifying emerging threats, leaving a time window of vulnerability [74]. Centralized databases such as Google Safe Browsing and Microsoft SmartScreen catalog known malicious URLs. These services process billions of URL checks daily, with Google Safe Browsing protecting approximately 4 billion devices worldwide [28].

Domain-Based Message Authentication (DMARC): This email authentication pro-

tocol combines Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM) to verify that messages originate from authorized servers. Organizations implementing DMARC report up to 80% reduction in domain spoofing attacks [23].

2.3.2 Client-Side Protection Technologies

Additionally, heuristic-based methods, which rely on predefined rules and visual features, offer some ability to detect novel attacks but often produce high false positive rates and degrade over time [82]. These solutions operate directly on user devices, providing an additional defense layer:

Browser-Based Security Features: Major browsers incorporate protective mechanisms including Safe Browsing integration, certificate validation, and suspicious site warnings. Chrome's Enhanced Safe Browsing feature claims to provide 35% better protection against phishing compared to standard protections [27].

Anti-Phishing Extensions: Browser extensions like Netcraft Anti-Phishing, Bitdefender TrafficLight, and PhishDetect offer specialized protection through various detection methods. These tools supplement native browser protections with additional heuristics and connection to specialized threat intelligence sources [83].

Password Managers: Beyond credential management, modern password managers incorporate phishing detection by verifying domain legitimacy before auto-filling credentials. This approach effectively counters sophisticated domain spoofing attempts [47].

2.3.3 Machine Learning and AI-Based Detection Approaches

More recently, machine learning-based approaches have shown promise by using lexical, structural, and behavioral features to identify suspicious content. For instance, ensemble models such as Random Forest classifiers can achieve high accuracy in laboratory environments [43]. However, these methods face practical challenges, including real-time performance constraints, limited availability of labeled data, and vulnerability to adversarial attacks. Artificial intelligence has transformed phishing detection capabilities:

Supervised Learning Models: These systems train on labeled datasets of legitimate and phishing examples to identify distinguishing characteristics. Common algorithms include Random Forests, Support Vector Machines, and Logistic Regression, achieving detection accuracies between 95-98% in controlled environments [45].

Natural Language Processing (NLP) Techniques: NLP models analyze linguistic patterns in communications to identify anomalies consistent with phishing attempts. Ad-

vanced systems detect subtle indicators including tonal inconsistencies, urgency markers, and grammatical patterns associated with non-native writers [61].

Computer Vision Approaches: These techniques analyze visual similarity between legitimate websites and potential phishing pages. By converting screenshots into feature vectors, these systems can identify visual impersonation attempts with up to 97% accuracy in experimental settings [89].

2.3.4 Human-Centric Defense Strategies

Furthermore, user training and awareness programs, while useful, are often ineffective under cognitive load or in the presence of convincing attack cues [46, 93]. Recognizing that technical measures alone cannot eliminate phishing risks, organizations increasingly implement human-focused countermeasures:

Security Awareness Training: Structured educational programs teach users to identify phishing indicators and appropriate response procedures. Meta-analyses of training effectiveness show properly designed programs can reduce susceptibility by 40-60% citeSANS2024.

Simulated Phishing Campaigns: Organizations deploy controlled phishing exercises to measure susceptibility and reinforce training concepts. Data from KnowBe4 indicates organizations conducting regular simulations experience a 65% reduction in click rates on actual phishing attempts over 12 months [42].

Just-in-Time Training: This approach delivers targeted educational content immediately after users engage in risky behaviors during simulated exercises. Research indicates this methodology improves retention by up to 40% compared to traditional training approaches [38].

Psychological Frameworks: Advanced training programs increasingly incorporate behavioral science principles to address the psychological vulnerabilities exploited by phishing. These approaches focus on countering cognitive biases like authority bias and urgency that facilitate successful attacks [94].

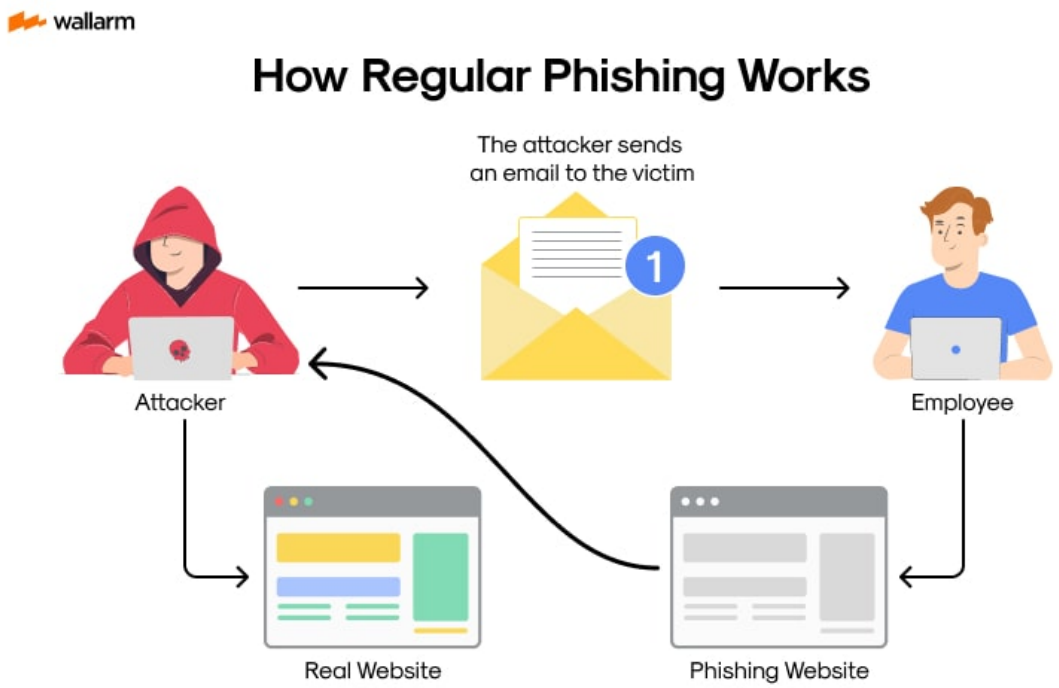


Figure 2.3: How regular phishing works

2.4 Browser Extension Solutions: Analysis and Evaluation

2.4.1 Market-Leading Browser Extension Solutions

One promising vector for defense is browser extensions, which operate at the client-side and have direct access to the user's browsing activity. Tools like Netcraft, Bitdefender TrafficLight, and PhishDetect integrate blacklist checks, heuristic rules, and community reports to block threats in real-time [11, 84]. Browser extensions represent a strategic defense layer against phishing, operating at the critical intersection between users and potentially malicious web content.

Netcraft Anti-Phishing Extension: Netcraft's extension represents one of the most established solutions in the browser security landscape, leveraging the company's extensive threat intelligence infrastructure. For example, Netcraft uses domain age and reputation scoring to detect a majority of phishing attempts. The detection methodology primarily relies on a centralized blocklist of known phishing URLs, supplemented by domain age checking and reputation scoring. The company claims processing over 5 million unique URLs daily [58]. Strengths include excellent performance against known threats with

minimal false positives (under 0.1% in independent testing). However, limitations show significantly reduced effectiveness against zero-day phishing sites, with detection rates falling to approximately 35-40% for newly created phishing infrastructure [22].

Bitdefender TrafficLight: This extension integrates with Bitdefender's broader security ecosystem, offering multi-layered protection. While Bitdefender combines heuristics with content analysis to achieve strong detection accuracy, it employs a hybrid approach combining cloud-based URL verification, heuristic scanning of page content, and behavioral analysis of scripts [15]. Strengths include strong performance against both established and emerging threats, with independent testing showing approximately 89% detection rates across all phishing categories. Limitations introduce noticeable performance overhead, increasing page load times by 15-20% on average [12].

PhishDetect: This open-source solution emphasizes privacy and transparency in its detection approach, implementing client-side heuristic analysis of DOM structures, focusing on identifying patterns common in phishing pages such as password fields, brand impersonation indicators, and suspicious JavaScript behaviors [68]. Strengths include operating without transmitting browsing data to external servers, addressing privacy concerns. Limitations show limited effectiveness against sophisticated phishing operations that employ evasion techniques [89].

2.4.2 Performance Analysis and Limitations

However, these tools have limitations such as performance degradation, reliance on static rules, limited cross-browser support, and poor handling of emerging threats [19]. Comprehensive evaluation of current browser-based anti-phishing extensions reveals several common limitations:

Static Rule Limitations: Most extensions rely heavily on predefined rules and patterns that fail to adapt to rapidly evolving phishing techniques. Analysis by [22] demonstrated that modification of just 4-5 key DOM elements allowed 76% of tested phishing pages to evade detection by leading extensions.

Delayed Detection of New Threats: Centralized blocklist approaches inherently create a protection gap between threat emergence and detection. Research indicates the average delay between phishing site deployment and inclusion in major blocklists ranges from 4 to 21 hours, providing a critical window of exposure [10].

Binary Classification Approach: Current solutions typically classify pages in a binary manner (safe/unsafe) without providing nuanced risk assessment or explanatory context. This approach increases user tendency to ignore or override warnings when false positives

occur [94].

2.4.3 Cross-Platform Compatibility and User Experience Issues

Limited Cross-Browser Compatibility: Most extensions exhibit inconsistent performance across different browser environments, with functionality often optimized for Chrome at the expense of Firefox, Safari, and Edge. This fragmentation creates security inconsistencies for users who employ multiple browsers [83].

Performance-Security Tradeoffs: Extensions implementing thorough security checks often introduce significant performance overhead, forcing users to choose between comprehensive protection and browsing experience. This compromise potentially leads to users disabling protection during performance-sensitive activities [12].

Inadequate Contextual Awareness: Existing solutions typically evaluate URLs or page content in isolation without considering the context of the user's navigation path or interaction patterns. This limitation reduces effectiveness against sophisticated attacks that exploit legitimate sites as part of multi-stage operations [44].

2.4.4 Effectiveness Assessment Against Real-World Threats

When measured against real-world phishing scenarios, current extensions demonstrate substantial detection gaps. Comparative testing conducted by [89] found detection rates against a corpus of 10,000 confirmed phishing URLs ranged from 67.8% to 91.4% across leading extensions, with performance declining sharply for threats less than 24 hours old. This performance analysis reveals that while existing browser extensions provide valuable protection, they exhibit significant gaps in coverage, particularly against emerging threats and sophisticated attack techniques. The limitations identified across leading solutions underscore the need for more adaptive, comprehensive approaches to browser-based phishing protection.

2.5 Research Gaps and Requirements for Enhanced Solutions

2.5.1 Identified Limitations in Current Approaches

Given the limitations of current solutions and the continued evolution of phishing methods, there is a clear need for an improved, adaptive browser-based tool that leverages real-time machine learning, URL and DOM analysis, threat intelligence integration, and efficient user interaction design. The identified limitations in existing anti-phishing extensions, combined with the continuously evolving threat landscape, establish a clear rationale for developing a new approach. The goal of such a solution should be to minimize false positives, reduce detection latency, maintain performance, and enhance cross-platform compatibility—offering not just protection, but also awareness through contextual feedback. Current solutions have not kept pace with the rapid evolution of phishing techniques, particularly those employing dynamic content generation, cloaking mechanisms, and multi-stage attacks. A purpose-built solution leveraging recent advances in machine learning and behavioral analysis can significantly narrow this innovation gap [45].

2.5.2 Integration Requirements for Comprehensive Protection

Integration of Complementary Detection Methods: Existing extensions typically rely too heavily on single detection methodologies, creating exploitable blind spots. A comprehensive approach integrating URL analysis, content inspection, behavioral monitoring, and threat intelligence would substantially increase detection coverage across the phishing spectrum [61].

Need for Improved User Experience: The frequent trade-off between security efficacy and usability in current solutions often results in user frustration and security fatigue. A carefully designed extension prioritizing both protection and seamless experience would improve adoption rates and sustained usage [94].

Opportunity for Educational Impact: Beyond mere blocking, a next-generation solution could incorporate contextual learning elements that improve users' security awareness during everyday browsing activities. This approach transforms security tools from passive protections into active educational platforms [38].

2.5.3 Technological Advancement Opportunities

Leveraging Advances in Machine Learning: Recent breakthroughs in lightweight ML models suitable for browser environments enable sophisticated detection capabilities previously impossible within performance constraints. These advances create opportunities for real-time analysis without sacrificing browsing experience [97].

Addressing Cross-Platform Inconsistencies: A modern solution designed with cross-browser compatibility as a foundational requirement would address the fragmentation issues prevalent in current offerings, providing consistent protection across users' increasingly diverse browsing habits [83].

2.5.4 Proposed Solution Architecture

The proposed browser extension will specifically target these identified gaps through a multi-layered detection architecture combining:

Machine learning-based URL analysis using optimized models capable of identifying suspicious lexical patterns and domain characteristics without requiring server connectivity.

DOM structure analysis focusing on identifying phishing patterns within page content, particularly login forms, brand impersonation attempts, and suspicious JavaScript behaviors.

Integration with authoritative external services such as Google Safe Browsing and VirusTotal to leverage global threat intelligence while maintaining privacy controls.

Contextual risk scoring rather than binary classification, providing users with graduated risk assessments and specific explanations for security warnings.

Performance-optimized architecture ensuring minimal impact on browsing experience through selective analysis techniques and efficient resource utilization. By addressing these critical aspects, the proposed extension aims to significantly advance the state of browser-based phishing protection, providing users with more effective, usable, and educational security tools than currently available alternatives.

Chapter 3

METHODOLOGY

3.1 Overview of the Methodology

This section describes the methodological steps taken to design and implement the browser extension "Safe Click – Phishing Prevention." The development process follows a quantitative approach, combining supervised machine learning, integration with external threat intelligence services, and content-based analysis of web pages. All components are evaluated using numerical data, statistical metrics, and experimental comparisons. The methodology was carefully chosen to balance accuracy, system efficiency, and ease of use, while adhering to contemporary cybersecurity and software development standards.

3.2 Dataset Description and Preprocessing

For this study, the "Phishing Website Detection Dataset" was selected as the primary data source. This dataset is one of the largest and most comprehensive open collections available for phishing detection research, with a total size of approximately 138 MB and 6,568,000 of unique URLs, each labeled as either benign or malicious.

The phishing URLs were collected from phishtank.org, ensuring a wide variety of real-world phishing techniques, while the benign URLs were sourced from commoncrawl.org, representing legitimate web traffic. The dataset is organized into separate folders for training and testing, each further divided into “malicious” and “benign” subfolders. This structure provides a clear separation of data for model development and evaluation.

The main reasons for choosing this dataset are supported by recent research and best

practices in the field:

- Scale: Large-scale datasets are essential for training robust and generalizable machine learning models in cybersecurity[41, 72].
- Openness: The dataset is freely accessible under a permissive license, supporting reproducibility and transparency in research, which is strongly recommended in the scientific community [96].
- Realism: The data is sourced from real-world web activity, including up-to-date phishing campaigns and a diverse set of benign URLs. Using real-world data is critical for developing models that perform well in practical scenarios [41, 72].
- Clear labeling: Each URL is explicitly labeled, facilitating supervised learning and reliable evaluation. Accurate labeling is highlighted as a key factor for effective phishing detection in multiple studies [9, 35].

To streamline the machine learning workflow, all URLs from the original folder structure were consolidated into a single CSV file (malicious_phish.csv). Each entry in this file contains a URL and its corresponding binary class label (0 for benign, 1 for malicious).

The final consolidated dataset contains approximately 3.1 million benign URLs (class 0) and 3.5 million malicious URLs (class 1), resulting in a nearly balanced class distribution. This balance is visually represented in Figure 3.1.

A balanced dataset is crucial for training unbiased machine learning models and is recommended in the literature to avoid skewed performance towards the majority class[41, 96]. The near-equal representation of both classes in this dataset ensures that the classifier can learn to distinguish between benign and malicious URLs effectively, supporting robust real-world deployment. This approach aligns with recommendations from recent surveys and empirical studies, which emphasize the importance of using large, diverse, and well-labeled datasets for developing effective phishing detection systems[41, 72, 96].

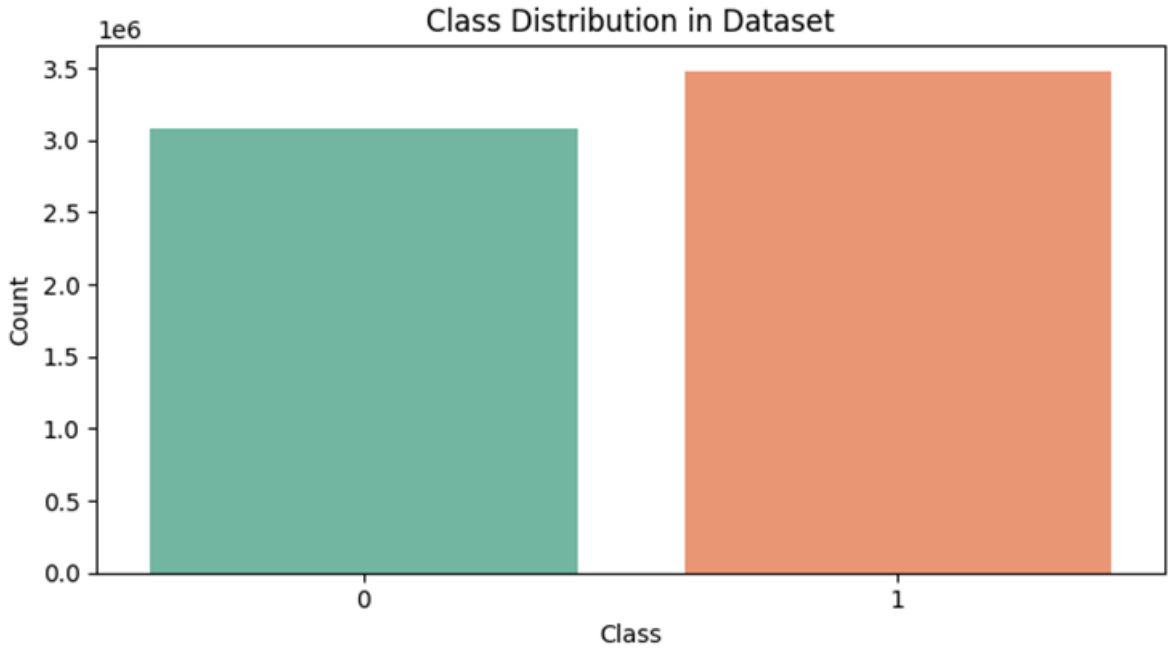


Figure 3.1: Class Distribution in Dataset

3.2.1 Preprocessing Steps

Preprocessing constitutes a critical phase in the preparation of data for machine learning models, particularly in the domain of phishing detection. The objective of this stage is to ensure that the dataset is clean, consistently formatted, and suitable for automated learning processes. As emphasized in prior research, well-structured preprocessing enhances both the accuracy and stability of classification models in cybersecurity contexts [50].

The preprocessing pipeline applied in this project consisted of three core stages: **consolidation and labeling**, **cleaning and balance verification** , and **feature extraction** based on URL structure.

3.2.1.1 Consolidation and Labeling

To facilitate supervised machine learning, all URLs from the original dataset structure were merged into a single CSV file (malicious_phish.csv). Each URL was assigned a binary class label:

- Benign → 0
- Phishing, defacement, malware → 1

This consolidation simplifies the data pipeline and ensures compatibility with standard machine learning workflows. The use of binary labeling is widely recommended in the literature for real-time phishing detection systems, as it streamlines both model training and evaluation [9, 34].

3.2.1.2 Cleaning and Balance Verification

After consolidation, the dataset underwent a cleaning process to remove duplicate and malformed URLs, ensuring data integrity and preventing potential bias during model training. Maintaining a clean dataset is essential for reliable machine learning outcomes, as highlighted in recent research [41, 50]. Following cleaning, the class distribution was analyzed to verify balance between benign and malicious samples. As shown in Figure 1, the dataset contains nearly equal numbers of both classes, which helps prevent model bias and supports robust evaluation [96]. This balance eliminates the need for additional resampling or class-weighting techniques.

3.2.1.3 Feature Extraction

For each URL, a set of 34 handcrafted features was extracted to capture structural, lexical, and semantic characteristics relevant to phishing detection. The selection of these features is grounded in their proven effectiveness in the literature and their practical relevance for distinguishing between benign and malicious URLs [26, 34, 41]. Feature extraction was performed dynamically using a custom script (extract_features.py), ensuring consistency and up-to-date logic throughout the project.

The features are categorized and described as follows:

url_length: Total length of the URL. Longer URLs often indicate obfuscation or attempts to hide malicious content[87].

domain_length: Length of the domain part. Short or random domains are common in phishing[35].

num_dots: Number of dots in the URL. Multiple subdomains can be used to mimic legitimate sites [1].

num_slashes: Number of slashes. Excessive slashes may indicate redirection or obfuscation.

num_hyphens: Number of hyphens. Hyphens are often used to create deceptive domains [14].

num_underscores: Number of underscores. Rare in legitimate domains, may indicate manipulation.

num_question_marks: Number of "?" characters. Used to introduce query parameters, sometimes for tracking or obfuscation.

num_equals: Number of "=" characters. Common in query strings, may indicate data submission.

num_at: Number of "@" symbols. Used to mask the real domain, a known phishing tactic[87].

num_and: Number of "&" characters. Used in complex query strings.

num_exclamation: Number of "!" characters. Rare in legitimate URLs, may indicate suspicious intent.

num_spaces: Number of spaces. Spaces are unusual in URLs and may indicate obfuscation.

num_tildes: Number of ~ characters. Sometimes used in personal or temporary pages.

num_commas: Number of commas. Rare in legitimate URLs.

num_plus: Number of "+" characters. Used in URL encoding or obfuscation.

num_asterisks: Number of "*" *characters. Rare, may indicate manipulation.*

num_hashes: Number of "#" characters. Used for anchors, sometimes abused in phishing.

num_dollars: Number of "$" characters. Rare, may indicate suspicious activity.

num_percent: Number of "%" characters. Used in URL encoding, sometimes for obfuscation.

num_special_chars: Total count of special characters. High counts may indicate obfuscation.

has_https: Whether the URL uses HTTPS. While not definitive, lack of HTTPS is a risk factor [26]

has_ip: Whether the domain is an IP address. Legitimate sites rarely use raw IPs[14].

has_port: Whether a port is specified. Unusual ports may indicate suspicious activity.

has_query: Whether the URL contains a query string.

has_anchor: Whether the URL contains an anchor ("#").

has_digits_in_domain: Presence of digits in the domain. Often used in automatically generated or deceptive domains.

suspicious_tld: Whether the top-level domain is known to be abused (e.g., .xyz, .tk, .info).

num_subdomains: Number of subdomains. Multiple subdomains can be used to mimic trusted brands.

is_shortening_service: Whether the URL uses a known shortening service (e.g., bit.ly, tinyurl). Shortened URLs are often used to hide malicious destinations [41].

suspicious_words_count: Count of suspicious keywords (e.g., "login", "secure", "account", "paypal"). Such words are frequently used in phishing URLs [26, 35].

path_length: Length of the path component. Unusually long or complex paths may indicate obfuscation.

query_length: Length of the query string.

domain_entropy: Entropy of the domain. High entropy may indicate randomness, typical for phishing domains.

path_entropy: Entropy of the path.

query_entropy: Entropy of the query string.

These features collectively capture a wide range of phishing indicators, from structural anomalies to lexical tricks and statistical irregularities. Their inclusion is supported by numerous studies demonstrating their effectiveness in automated phishing detection[14, 26, 35, 41].

3.3 Algorithm Selection and Comparative Evaluation

3.3.1 Algorithms Evaluated

In this study, four supervised machine learning (ML) algorithms were selected and evaluated for the binary classification task of phishing URL detection:

- Random Forest (RF)
- Logistic Regression (LR)
- Decision Tree (DT)
- Linear Support Vector Machine (Linear SVM)
- CatBoost
- Neural Network (NN)

The choice of these algorithms was motivated by their varied learning mechanisms, their effectiveness in working with structured tabular data, and their frequent application in cybersecurity-related classification tasks[6, 41, 72]. Among them, Random Forest is particularly recognized for its resistance to overfitting, capacity to handle high-dimensional inputs, and built-in support for estimating feature importance, making it a strong candidate for phishing detection in dynamic environments [50].

3.3.2 Evaluation Procedure

To ensure the fairness and reproducibility of the comparative evaluation, the dataset was divided into training and testing sets using an 80/20 stratified split, preserving the class distribution in both subsets. Stratification is especially important in phishing detection due to class imbalance between benign and malicious samples [96].

Prior to model training, all features were standardized using **StandardScaler** to normalize their distribution. This preprocessing step is particularly important for algorithms such as SVM and KNN, whose performance can be significantly affected by differences in feature scale [26, 35]. The performance of each model was assessed based on a set of commonly used classification metrics, applied to the test data:

- Accuracy – measures the overall proportion of correct predictions across both classes.
- Precision – reflects the proportion of correctly identified malicious URLs among all URLs classified as malicious. This is especially important for reducing false positives.
- Recall – indicates the model's ability to correctly detect phishing URLs, minimizing false negatives.
- F1-score – provides a balanced measure by combining precision and recall, particularly useful in scenarios with class imbalance.
- ROC-AUC – evaluates the model's ability to distinguish between classes across all threshold levels, offering a more comprehensive view of discriminatory power.

These metrics were chosen to ensure a multidimensional evaluation of model performance, balancing correctness, sensitivity to phishing detection, and computational cost — all critical factors for real-time applications such as browser-based phishing prevention[6, 35].

3.3.3 Results and Model Selection

Figure 3.2 presents the performance of **Random Forest, Logistic Regression, Decision Tree, Linear SVM, CatBoost, and Neural Network** classifiers across five key metrics: **accuracy, precision, recall, F1-score, and ROC-AUC**. As shown, the Random Forest model consistently achieves the highest scores across all evaluation metrics, indicating superior classification performance and generalizability. Logistic Regression, Decision Tree, Linear SVM, CatBoost, and Neural Network also demonstrate strong results, but with slightly lower recall and F1-scores compared to Random Forest. These findings support the selection of Random Forest as the final model for deployment in

the phishing detection system, in line with prior research highlighting its robustness and effectiveness for cybersecurity applications [6, 41].

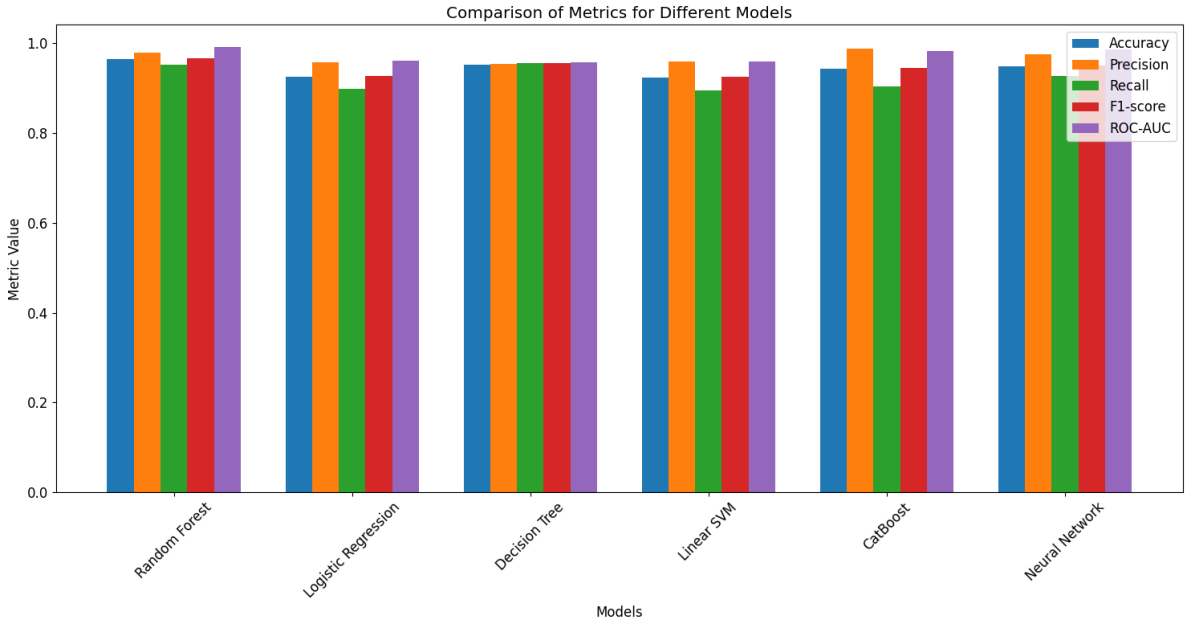


Figure 3.2: Model Comparison

The table 3.1 and Figure 3.3 report the accuracy, precision, recall, F1-score, and ROC-AUC for all evaluated classifiers. Among all models, the Random Forest algorithm demonstrates the highest overall performance, achieving an accuracy of 0.965, precision of 0.980, recall of 0.953, F1-score of 0.966, and ROC-AUC of 0.992. Other models also show competitive results, but none surpass Random Forest across all key metrics. These findings confirm the suitability of Random Forest for real-time phishing detection, as it provides the best balance between precision and recall, and achieves the highest discriminatory power as measured by ROC-AUC. This result is consistent with previous research, which highlights the effectiveness of ensemble methods for cybersecurity applications [6, 41].

№	Model	Accuracy	Precision	Recall	F1-score	ROC-AUC	Train Time (s)
0	Random Forest	0.965	0.980	0.953	0.966	0.992	142.20
1	Logistic Regression	0.925	0.957	0.899	0.927	0.961	44.89
2	Decision Tree	0.953	0.955	0.956	0.955	0.958	50.07
3	Linear SVM	0.924	0.959	0.895	0.926	0.959	804.23
4	CatBoost	0.944	0.988	0.905	0.945	0.983	90.00
5	Neural Network	0.949	0.976	0.927	0.951	0.986	1243.74

Table 3.1: Performance comparison of machine learning models on the phishing URL detection task.

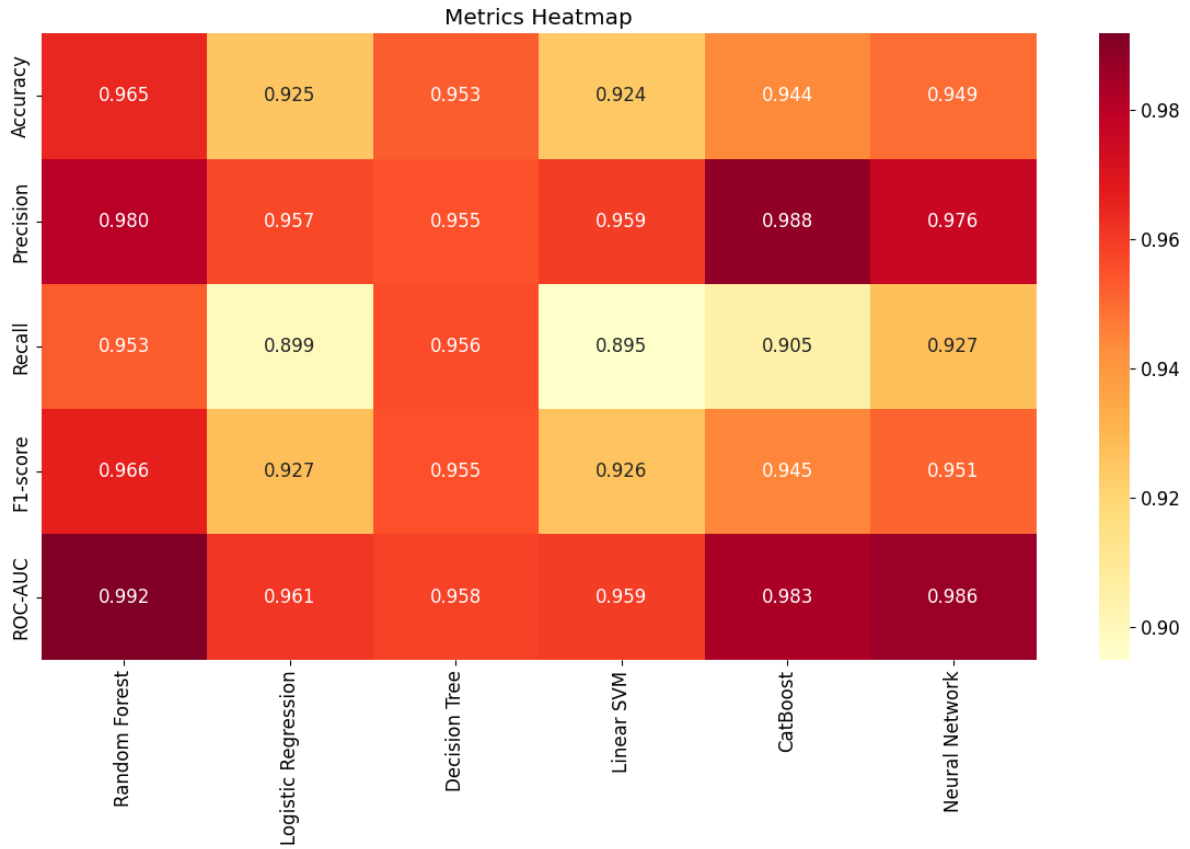


Figure 3.3: Metrics Heatmap

Given these comparative results, it became evident that machine learning not only outperforms traditional rule-based approaches but also provides the flexibility and scalability necessary for modern phishing defense. This naturally leads to the rationale for adopting machine learning in this project, both as a methodological foundation and as a core element of the system's architecture.

3.3.4 Rationale for Using Machine Learning

Traditional phishing detection methods, such as blacklists and heuristics, are constrained by their inability to detect novel or slightly modified threats [72, 98]. In contrast, machine learning models learn from data and can generalize to previously unseen patterns, making them more effective in dynamic environments.

In this project, Random Forest was chosen for its balance between accuracy, robustness, and interpretability. It handles high-dimensional data well and provides feature importance metrics, which are useful for both model transparency and security analysis [49, 50].

3.4 Integration of External Security APIs

To improve detection reliability and minimize false negatives, the system incorporates external threat intelligence services—namely Google Safe Browsing (GSB) and VirusTotal (VT)—as supplementary verification mechanisms. These services provide continuously updated information based on large-scale threat databases and are widely recognized for their effectiveness in phishing detection [54, 91]. Their integration complements the predictions made by the ML-based classifier and provides cross-verification through reputable external sources.

3.4.1 Selection and Description of External APIs

Google Safe Browsing is an industry-standard API maintained by Google. It provides near real-time classification of URLs based on their association with various threat categories, including *malware*, *social engineering*, *unwanted software*, and *potentially harmful applications*. In the implemented system, GSB is invoked when a URL is classified as benign by the ML model, serving as a second-layer defense to detect known phishing sites not captured during training[28, 91]

VirusTotal, a platform owned by Chronicle (a Google subsidiary), aggregates the results of over 70 antivirus engines and provides a reputation score for each URL. The system interprets a URL as suspicious if fewer than 95% of engines classify it as clean (i.e., confidence_score < 0.95). This strategy aligns with existing research advocating ensemble methods that combine local classifiers with third-party reputation services to reduce false negatives [54].

The selection of GSB and VT was guided by several practical and technical considerations:

- **Reputation and Validation in Literature**: Both services are widely adopted in academic studies and production systems. GSB is integrated into major browsers such as Chrome, Firefox, and Safari, while VT has been cited for its utility in phishing detection research[55].
- **Free Access and Prototyping Feasibility**: Public tiers of both APIs allow for non-commercial usage, which is suitable for academic research and prototyping.

- **Ease of Integration**: Each service offers RESTful API endpoints with JSON-formatted responses, along with extensive documentation. This makes them readily integrable into the Flask-based backend using standard Python libraries[99].

By integrating these two services, the system benefits from a layered detection strategy that combines learned patterns (via ML) with real-time threat intelligence from reliable external sources—enhancing overall detection robustness and minimizing the risk of undetected phishing attempts.

3.4.2 Benefits and Limitations of API Integration

Benefits	Limitation
Enhanced Accuracy APIs like GSB and VT can detect known malicious URLs missed by the ML model due to limited training data [54].	**Rate Limits** Free-tier APIs impose usage quotas, limiting scalability in high-traffic use [99]. GSB allows ~10,000 requests/day; VT public API permits ~4 requests/minute [88].
Cross-Validation Discrepancies between ML and APIs are flagged for user attention, enabling conservative risk assessment [91].	**False Negatives** APIs may still miss newly emerging threats or zero-day phishing domains [91].
Scalability APIs operate in the cloud and do not consume local resources, supporting real-time applications [55].	**Latency** Typical response time is <500 ms for GSB and ~1–2 seconds for VT, which may delay result delivery in some cases [55, 88].
Defense in Depth Layered detection improves robustness against diverse attack vectors [98].	**Dependency on Internet** APIs require active network connectivity; unavailable services may fail silently [99].

Table 3.2: Comparison of Benefits and Limitations of APIs

The integration of external threat intelligence APIs strengthens the phishing detection pipeline by providing reputation-based verification, complementing the predictions of the local machine learning model. As summarized in the table above, services such as Google Safe Browsing and VirusTotal improve detection accuracy, enable cross-validation of results, and scale well in real-time environments. However, their effectiveness is accompanied by practical constraints, including API rate limits, response latency,

and dependency on internet connectivity. For instance, Google Safe Browsing typically responds in under 500 milliseconds with phishing and malware coverage, while VirusTotal offers broader detection via 70+ antivirus engines but at a slower rate and with stricter free-tier limits [88, 99].

These limitations are addressed through mitigation strategies implemented at the architectural level—such as response caching, layered decision logic, and fallback handling—to preserve both responsiveness and robustness. This design approach aligns with best practices in phishing detection literature that emphasize fault-tolerant, multi-layered systems capable of adapting to evolving threats[54, 91].

3.5 DOM Analysis of Web Pages

Beyond analyzing URLs with machine learning models and external security APIs, the system includes an additional content-level verification layer based on the Document Object Model (DOM). This inspection targets the structural and semantic composition of web pages, allowing for the identification of phishing attempts that may not be detectable through URL-based analysis alone. Such layered evaluation has proven especially useful for identifying threats that mimic legitimate layouts while bypassing conventional URL checks.

3.5.1 Heuristic Features Used in DOM Analysis

The module applies several heuristics, drawn from existing literature and security standards, to identify potentially malicious behavior embedded in the page's Document Object Model. For example, the system checks for:

- **Presence of login forms** — specifically <input type="password"> fields, which often indicate attempts to capture user credentials. This approach is recommended by OWASP and has been widely adopted in phishing detection systems [35, 65].
- **Suspicious keywords** in page content, such as "login", "verify", "account", or "secure", which are known to simulate urgency or authority and influence user behavior[59].

- **Deceptive form behavior**, including form actions pointing to third-party domains, use of insecure GET methods, or disabling of autocomplete — all of which may indicate phishing intent.
- **Mismatch between page origin and form submission destination**, suggesting redirection to an external source, a typical sign of credential theft attempts [35].

Research confirms the importance of DOM analysis as a complementary detection mechanism. Nguyen et al. demonstrate that phishing pages frequently replicate the structure of legitimate websites while altering their functional components[59]. Similarly, Reynolds and Rainear highlight how domain mimicry and layout deception can successfully bypass URL-only filters, making content-level analysis essential[71].

This DOM-based approach allows the system to detect sophisticated phishing strategies that are not always reflected in URL structures alone, enhancing protection for end users in real-time browsing scenarios.

3.5.2 Rationale and Design Considerations

The selection of features for DOM inspection was informed by real-world phishing behavior and recommendations found in industry literature. Each heuristic was chosen based on its empirical relevance and practical feasibility in browser-based environments.

- **Empirical Foundation**: All checks—such as detection of login forms, suspicious keywords, and external submission links—are grounded in patterns observed in actual phishing pages and have been validated in research [59, 65].
- **Lightweight Execution**: The heuristics are implemented using standard JavaScript functions within the browser's content script. This ensures that DOM analysis introduces minimal performance overhead, making it suitable for real-time scanning[59].
- **Zero-Day Detection Capability**: Unlike blacklists or machine learning models trained on static datasets, DOM-based inspection can flag previously unknown threats by analyzing behavioral cues and page structure irregularities[78].

3.5.3 Role and Benefits of DOM Analysis

The integration of DOM inspection plays a complementary role alongside machine learning and reputation-based APIs. While ML models and services such as Google Safe Browsing focus primarily on URL-level and historical reputation signals, DOM analysis provides a content-aware layer of defense.

Phishing sites often replicate the layout and visual elements of legitimate services. By examining the presence of suspicious forms, the use of deceptive field configurations, and keyword indicators, the system can identify impersonation attempts that would otherwise evade URL-based classifiers [71].

In addition to improving detection rates, the DOM analysis module promotes user awareness. When a threat is detected, the extension provides clear feedback regarding which page elements were considered suspicious. This approach not only improves transparency but also helps users better understand phishing techniques, reinforcing security awareness [65, 71].

3.5.4 Implementation Overview

The DOM analysis logic is fully contained within the browser extension's content.js script. It operates through a modular structure:

- Each heuristic (e.g., login form detection, keyword matching, form action inspection) is encapsulated in a dedicated function.
- The results of the analysis are returned as a structured list of triggered heuristics.
- This list is sent to the popup interface and displayed in the "DOM Analysis" section when the user initiates a scan.

This diagram illustrates a common flow in a DOM-based phishing attack. The sequence begins when an attacker crafts a malicious URL (Step 1), often containing embedded JavaScript or URL parameters that exploit browser behavior [59]. When the user clicks the deceptive link (Step 2), a legitimate or compromised website responds (Step 3). Malicious scripts then manipulate the Document Object Model (DOM) within the browser (Step 4), executing without user consent (Step 5). These scripts typically

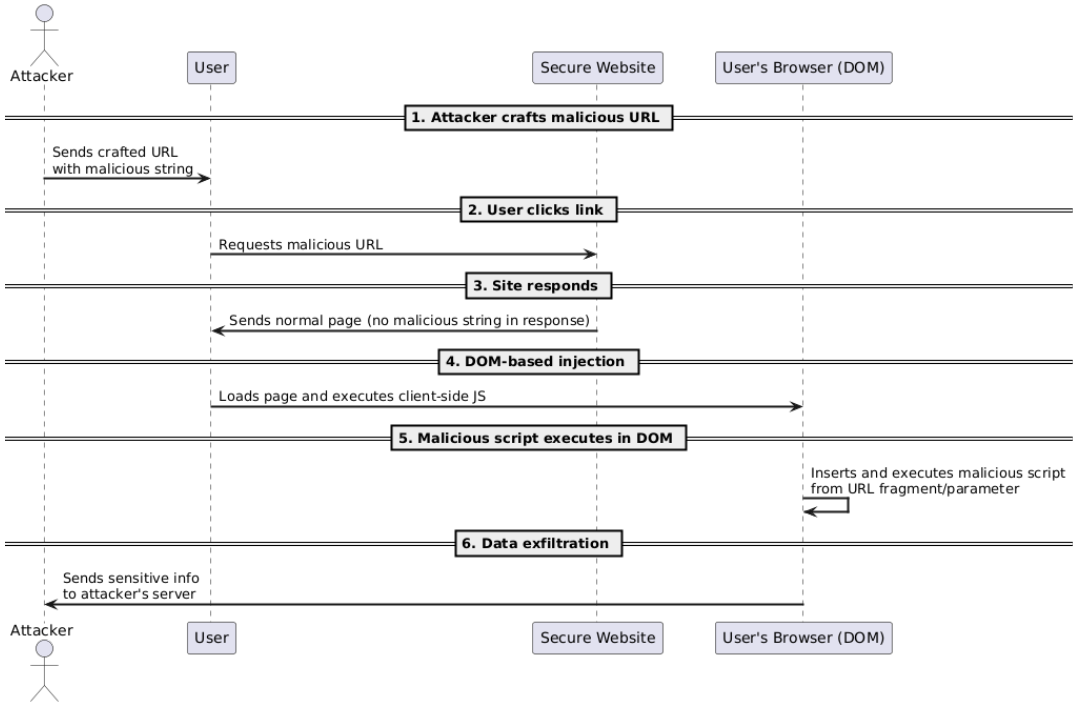


Figure 3.4: Sequence Diagram of a Typical DOM-Based Phishing Attack

aim to harvest sensitive information such as login credentials, which are exfiltrated to attacker-controlled servers (Step 6) [65, 71]. This form of attack is difficult to detect using URL blacklists alone. Therefore, browser-side countermeasures—such as real-time DOM analysis and behavioral detection—are critical [59, 65].

DOM-based cross-site scripting

To minimize performance impact and give users control, DOM inspection is executed manually on demand. This design balances responsiveness with user experience and aligns with OWASP guidelines for phishing detection in client-side environments[65].

3.6 Rationale for Choosing Google Chrome and a Browser Extension

3.6.1 Selection of Google Chrome as the Deployment Platform

The decision to develop and deploy the phishing detection system specifically for the Google Chrome browser stems from its widespread adoption and technical advantages.

According to recent global statistics, Chrome remains the most commonly used browser across platforms, offering the highest user coverage for any browser-based solution [80]. Given this extensive reach, developing a Chrome extension ensures the greatest potential impact for end-user protection.

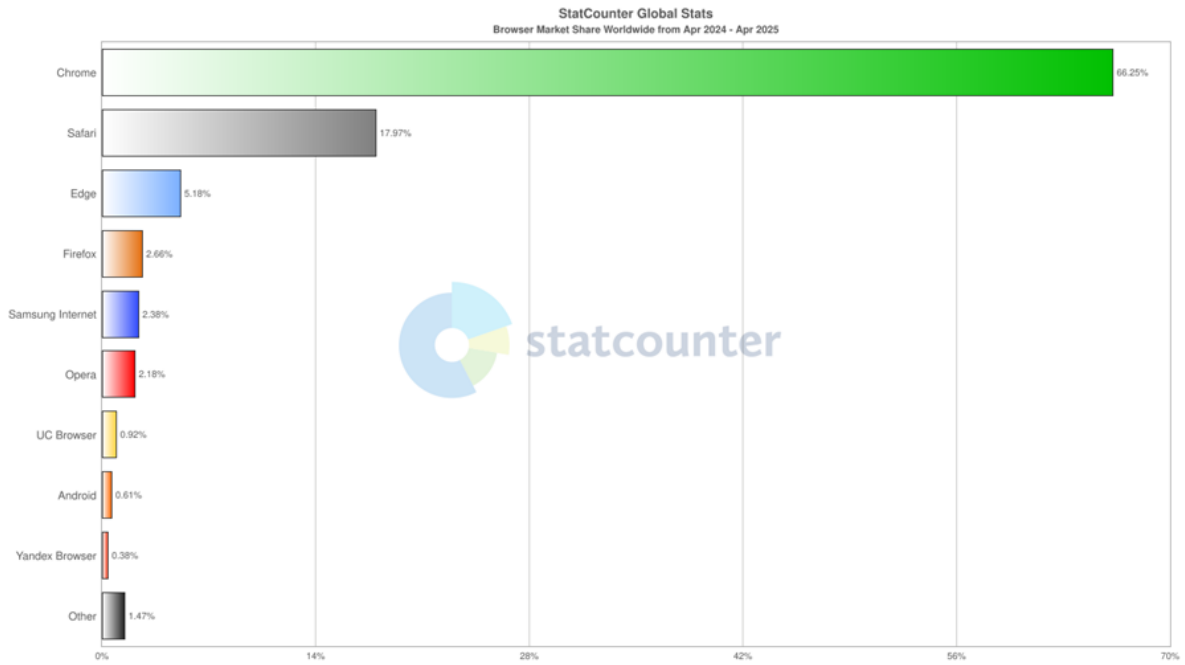


Figure 3.5: Browser Market from Apr 2024 – Apr 2025 [80].

Chrome's extension ecosystem is mature and feature-rich, providing extensive APIs for tab management, network request interception, secure storage, and direct DOM interaction. These capabilities are essential for implementing real-time phishing detection at the point of page load. Furthermore, the adoption of **Manifest V3** has improved the security architecture of Chrome extensions by enforcing stricter content policies and restricting potentially dangerous behaviors such as remote code execution. These security enhancements are well documented in recent evaluations of browser extension platforms[25].

Another factor that influenced the platform choice is Chrome's well-integrated distribution model via the **Chrome Web Store**, which supports automated updates and review workflows. This system facilitates fast deployment of updates and security patches without requiring user intervention—an essential attribute for maintaining up-to-date threat protection[75].

3.6.2 Rationale for Implementing a Browser Extension

Developing phishing protection as a browser extension, rather than as a standalone application, provides several distinct advantages, both technical and user-facing.

First, **real-time contextual analysis** is best achieved within the browser environment itself. Extensions can monitor user activity on visited pages and inspect elements such as form fields, hyperlinks, and page metadata in real time—an approach shown to be effective in detecting in-session phishing attacks[25].

Second, **usability and accessibility** play an important role. Browser extensions can be installed in a single step, often directly from the browser's marketplace, and are automatically updated in the background. This greatly reduces the barrier to adoption and ensures users are continuously protected without requiring manual updates[75].

Third, the extension model offers **tight UI/UX integration** with the browser. Pop-up notifications, context menus, and inline overlays can be used to communicate risks clearly and non-intrusively. These interaction mechanisms are preferred for delivering security feedback, as they are less likely to disrupt the user's workflow while still maintaining visibility [40].

Finally, Chrome extensions built with Manifest V3 are **compatible across multiple Chromium-based browsers**, including Microsoft Edge, Brave, and Opera. This cross-platform compatibility allows the solution to reach a broader audience without significant changes to the codebase[75].

In summary, choosing Chrome as the platform and the browser extension model as the deployment strategy enables a balance between security, usability, and scalability—key factors in the practical deployment of phishing prevention tools.

3.6.3 Security Trade-offs and Mitigation Strategies

While browser extensions offer significant advantages in terms of real-time, in-context phishing detection, they also present potential security risks if not properly designed. Prior research has shown that misconfigured permissions, excessive access rights, or the inclusion of remote scripts can render browser extensions vulnerable to exploitation [25].

To address these concerns, the proposed system follows current best practices in secure

extension development. Specifically, the extension requests only minimal permissions required for operation, such as activeTab, storage, and scripting. Moreover, sensitive computations—including feature extraction and URL classification—are executed on a locally hosted Flask server, thereby avoiding unnecessary exposure of data within the extension's runtime environment. This approach aligns with the recommendations put forth in recent audits of browser extension ecosystems, which emphasize permission minimization, modular architecture, and strict separation of concerns[25, 75].

3.6.4 Modular Architecture and Component Interaction

The system is designed following a modular client-server architecture that integrates three complementary phishing detection layers: URL-based machine learning, external API verification, and heuristic-based DOM analysis. Such a layered configuration has been shown to improve robustness and detection rates in phishing prevention systems [41, 59].

At a high level, the architecture consists of the following components:

- **Browser Extension (Frontend):** Handles user interactions and displays the result via a popup interface. It also initiates scanning processes by collecting the active tab's URL and triggering DOM inspection.
- **Background Script:** Acts as an intermediary between different components of the extension (popup, content, and backend). It coordinates message passing and handles API call sequencing.
- **Content Script:** Injected into every visited page, this script inspects the DOM for suspicious elements such as login forms, keyword usage, or unsafe form behaviors. It reports its findings back to the popup.
- **Flask-Based Backend Server:** Serves as the processing hub. It extracts handcrafted features from the received URL, executes the pre-trained Random Forest model, and calls external APIs (Google Safe Browsing and VirusTotal) for additional verification.
- **Machine Learning Module:** A trained Random Forest classifier that produces binary predictions and risk scores based on extracted URL features.

- **API Integration Module:** Queries Google Safe Browsing and VirusTotal for real-time threat intelligence, complementing the ML decision with externally validated reputation data.

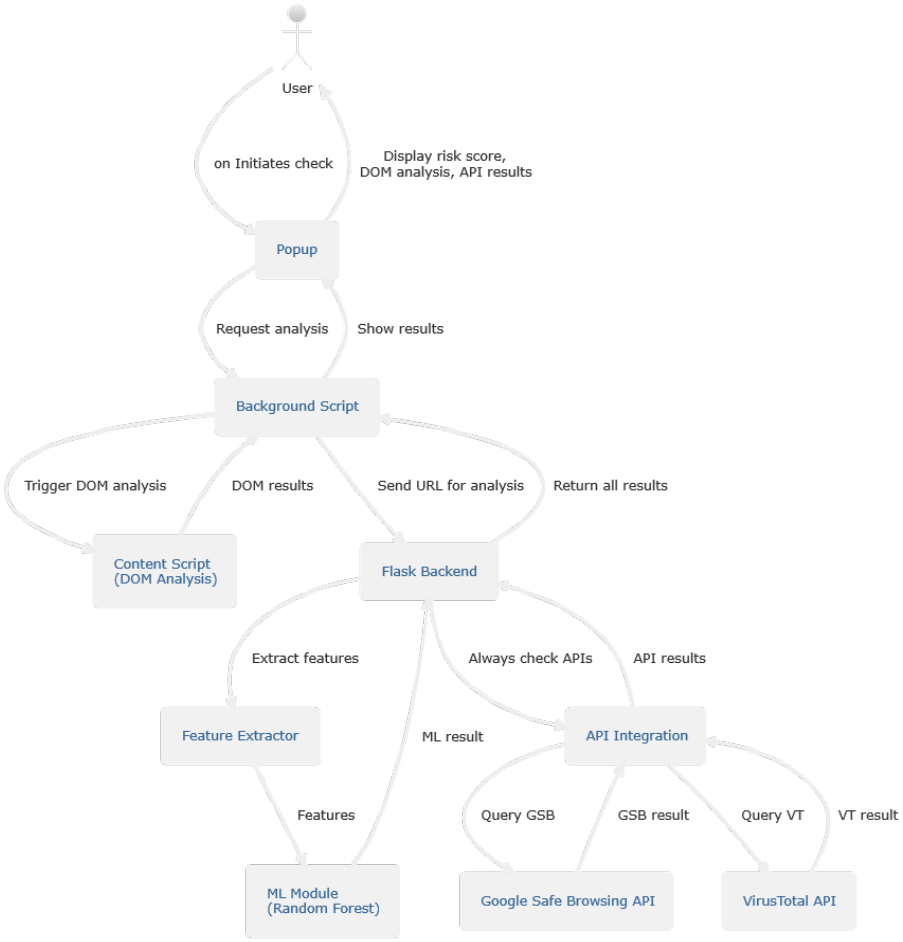


Figure 3.6: System Architecture and Component Interaction Flow of Developed Extension

This modular design enables each component to be maintained, tested, or updated independently, which is crucial for scalability and long-term sustainability in browser-based security tools[54].

To support seamless interaction between the browser extension and the backend, the system uses a RESTful communication protocol. The extension sends URL data via HTTP POST requests, while the server returns a structured JSON response containing classification results and external API outputs. This approach follows modern standards in web-based security systems and allows for straightforward extension and debugging [99].

The overall data flow and interaction between these components are illustrated in Figure 3.6. This diagram provides a visual representation of how user actions in the browser interface initiate a series of coordinated processes across the extension's frontend, backend, and external APIs. It clearly outlines the modular communication structure, emphasizing the roles of the popup interface, content and background scripts, and the Flask backend server. By mapping out the full sequence — from DOM inspection and URL classification to external API validation — the diagram reinforces the system's layered design and its adherence to separation of concerns, as recommended in contemporary cybersecurity architectures [41, 59].

3.7 User Feedback Assessment

To evaluate the practical usability and perceived effectiveness of the developed browser extension, a structured user survey was conducted. This step aimed to gather qualitative and quantitative feedback on the system's performance from end users after the extension was demonstrated and explained.

A Google Forms questionnaire was developed, comprising 15 questions, including multiple-choice and Likert-scale items. The questions focused on several core areas: perceived accuracy of phishing detection, clarity of the user interface, confidence in warnings, and overall user satisfaction with the extension. The survey design followed usability evaluation principles as recommended by ISO 9241-11 standards [33] and the widely recognized usability heuristics by Nielsen [60].

The survey was distributed to a **sample of 30 voluntary participants**, hereafter referred to as a **general digital user group**. The sample included students and early-career professionals from both technical and non-technical backgrounds, representing typical browser users. This diversity was intended to reflect a realistic cross-section of internet users and ensure feedback applicability across varied user profiles [62].

The data collection process respected user privacy: all responses were anonymous, and no personally identifiable information was gathered. This procedure aligns with ethical standards for human-subjects research [16]. The aggregated results are analyzed and discussed in Chapter 4 (Results and Discussion), providing additional validation for the system's design decisions, usability, and real-world applicability.

3.8 Summary of Methodology

The methodology outlined in this chapter describes a comprehensive and layered approach to phishing detection, integrating supervised machine learning, in-browser structural analysis, and external reputation services. Each component was selected based on established practices in the cybersecurity domain and validated through current academic literature.

The use of the Random Forest algorithm enables accurate binary classification while offering interpretability through feature importance scores. The incorporation of Google Safe Browsing and VirusTotal APIs adds redundancy by cross-referencing predictions with authoritative external databases, thereby enhancing overall reliability. In parallel, the DOM analysis module strengthens detection capabilities by identifying suspicious elements embedded in the structure and behavior of web pages—particularly useful for recognizing zero-day threats and impersonation attempts.

The system's modular architecture allows for individual components to be updated or extended independently, supporting long-term adaptability. Communication between components is handled through a RESTful API interface, promoting scalability and ease of integration with additional services or models in the future.

By implementing this detection pipeline within a Google Chrome extension, the solution delivers real-time, context-aware protection directly within the user's browsing environment. This integration maximizes usability while ensuring timely threat mitigation at the point of user interaction [59, 65]. The design emphasizes transparency, reproducibility, and compatibility with future academic or commercial enhancements.

Chapter 4

PRACTICAL PART

4.1 Implementation

4.1.1 Introduction

As a component of this final qualification project, a complete system has been designed and realized for detecting phishing URLs. The system relies on the combination of a client-side Google Chrome extension and a server-side part implemented by means of the Python programming language and the Flask web framework. The main objective of the developed system is the detection of potentially hazardous (phishing) web pages and the presentation to users of an informative interface for establishing the level of risk upon clicking on links. Machine learning techniques in conjunction with external analysis services are employed in the system for enhancing the validity and reliability of the results. The system is parameterizable and offers graphical visualization of the results in a friendly format.

4.1.2 System Architecture

The system designed has a two-part architecture (as evident from Fig. 4.1), consisting of a client part as a browser extension and a server part. The extension (frontend) is responsible for handling user interaction: it monitors the current URL of the active browser tab, sends requests for analysis, shows the safety status of the website to the user, and offers additional functionality (manual URL check, history log, and security settings). The server module is a Flask-based RESTful web service that takes requests from the extension, does URL feature extraction using a special processing module (`utils.py`),

preprocesses the data, and utilizes a pre-trained machine learning model to predict the phishing probability. External APIs are integrated into the server code; for example, the system checks URLs against Google Safe Browsing and VirusTotal to make the analysis more valid by cross-referencing with known malicious sources. Exchange of data between the server and the extension is conducted in JSON format using the HTTP protocol. Figure 4.1 schematically shows the interaction of the system components: the extension sends a URL for analysis, and the server responds with a structured answer containing the computed risk (as percentage) and additional information (e.g., flags of external services). Based on the feedback from the server, the extension either displays something to the user or takes some protective action (e.g., blocks the loading of a malicious site).

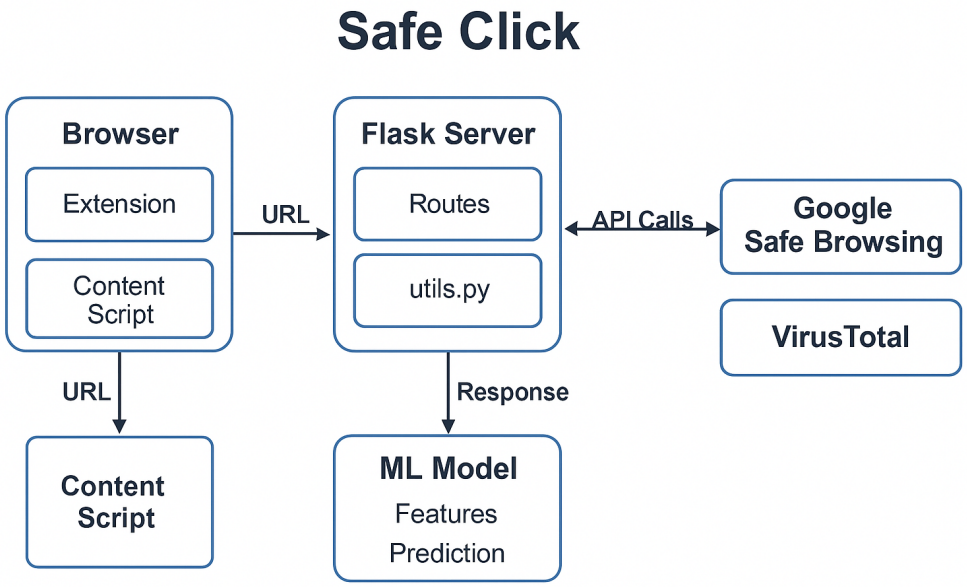


Figure 4.1: System architecture showing communication between browser extension and backend service

4.1.3 Popup Interface of the Extension.

The extension's user interface is realized as an HTML5, CSS3, and JavaScript popup window. When the browser icon of the extension is clicked, a popup window (`popup.html`) appears with the primary control elements. The interface is split between four tabs, quite evenly representing the system's functionality: **Current Site** (current website analysis), **Check URL** (manual checking of an input URL), **History** (record of past checks), and **Settings** (security options). Figure 4.2 illustrates a typical **Current Site** tab window

with the output of active page analysis as color-coded threat levels (green, yellow, or red background based on threat level) along with a text status message (e.g., "Safe" for safe sites or a warning for suspect ones).

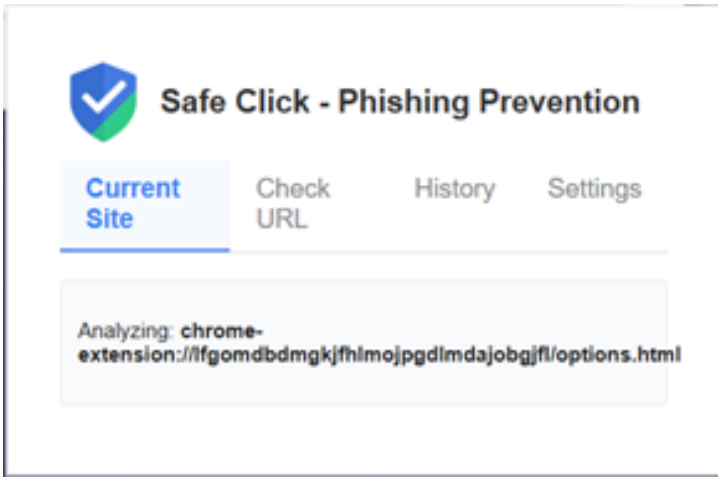


Figure 4.2: Current Site tab with threat level status

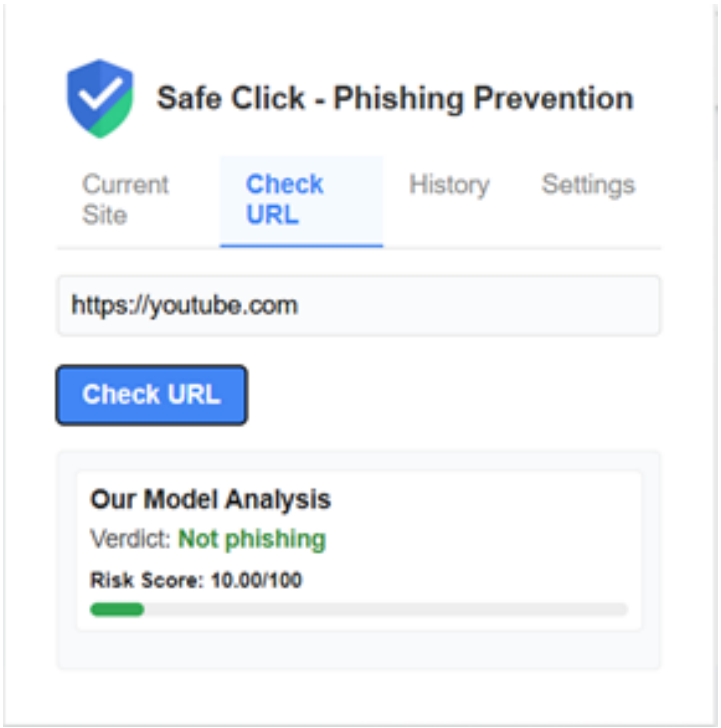


Figure 4.3: Extension settings window with thresholds and toggleable options

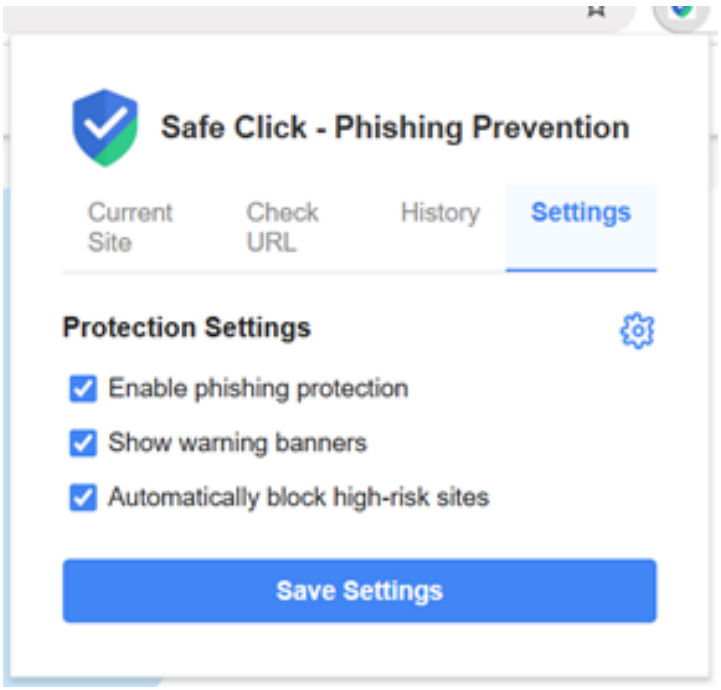


Figure 4.4: Extension settings window with thresholds and toggleable options

The **History** tab presents a record of previously scanned URLs with date, result, and risk level noted (see Fig. 4.5), and this data is retained locally via Chrome Storage. The **Settings** tab permits the user to establish threshold values for risk levels, turn some checks on or off (e.g., utilization of external APIs), and establish the system behavior (e.g., automatic blocking of malicious sites).

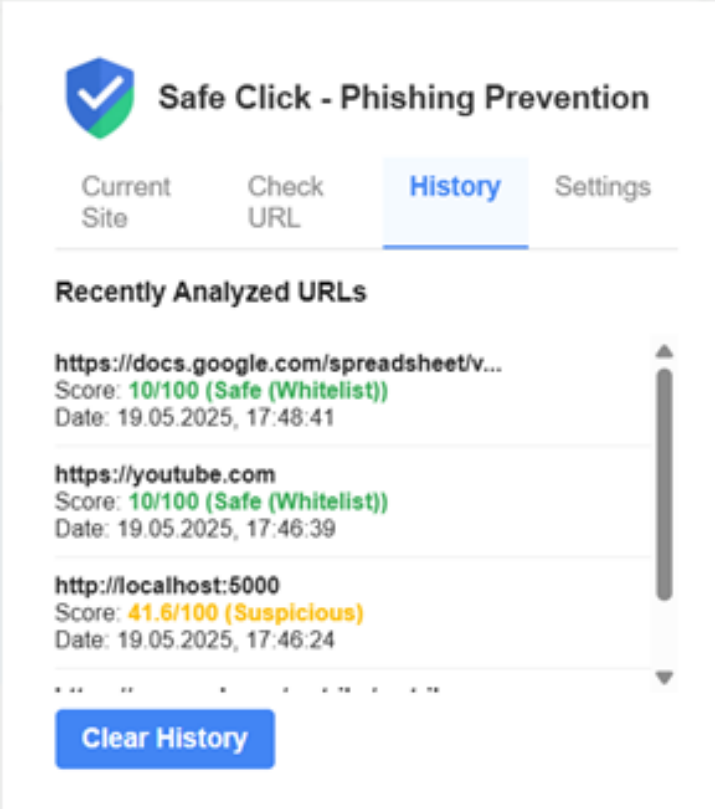


Figure 4.5: History tab showing previous URL checks

JavaScript scripts (`popup.js`) dynamically modify the popup content. They manage events like tab switch, loading the current page, and button click. When the extension popup is called on the **Current Site** tab, the script gets the active tab's URL and shows the previously loaded analysis result. The **Check URL** tab enables users to enter any URL manually: upon entering the address and clicking on the "Check" button, the extension makes a request to the server and shows the response in the popup. Below is a part of the popup script code that realizes the request to the server upon manual URL check and processes the response (Figure 4.6).

```
// Form element and status output area
const urlInput = document.getElementById('urlInput');
const checkBtn = document.getElementById('checkButton');
const statusText = document.getElementById('status');

// Handler for the "Check URL" button
checkBtn.addEventListener('click', () => {
    const url = urlInput.value.trim();
    if (!url) return;

    // Send a POST request to the Flask server for URL analysis
    fetch('http://localhost:5000/predict', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ url: url })
    })
    .then(response => response.json())
    .then(data => {
        // Update the interface based on the received result
        if (data.phishing) {
            statusText.textContent = Phishing detected! Risk:
            ↪  ${data.risk_score}%;
            statusText.style.color = 'red';
        } else {
            statusText.textContent = Safe. Risk: ${data.risk_score}%;
            statusText.style.color = 'green';
        }
        // Save the result in the local history log
        saveToHistory(url, data);
    })
    .catch(error => {
        statusText.textContent = 'Error: unable to check URL';
        statusText.style.color = 'gray';
        console.error('Check URL failed', error);
    });
});
```

Figure 4.6: Fragment of `popup.js`: Handling Manual URL Check and Displaying the Result

As Figure 4.6 shows, the popup script starts a fetch request against the REST API of the server, passing the URL to validate as JSON. Upon receiving the JSON response, the script analyzes the result fields (`data.phishing` – a flag indicating whether the site is phishing, and `data.risk_score` – the numerical risk score in percentage) and dynamically updates the interface: the `statusText` field displays a safety message and is colored green for safe URLs or red for phishing detections. The result is also saved to history (via the `saveToHistory` function), which inserts the entry into the browser's local storage and updates the **History** tab.

Content Script

The content script (`content.js`) is a part of the extension that is injected directly into the user's pages (according to the extension's manifest configuration). Its primary role is to automatically retrieve the URL of the current page and trigger server-side verification on the user's behalf. It can, if necessary, also alter the page content to safeguard the user.

In the developed extension, the content script is executed on all pages that are loaded

(for all URLs except trusted domains in a whitelist). The script gets the URL of the current page via `window.location.href`, sends an HTTP request to the Flask server (with the URL to check), and awaits the response.

The script handles the received risk level as follows: if the risk is low (less than 40 out of 100), nothing else is performed but logging the result; if the risk is medium (40–69), the script may warn the user by displaying a banner or a popup message above the page; if the risk is high (70 or more), active measures are taken, e.g., blocking content loading or redirecting the user to the warning page.

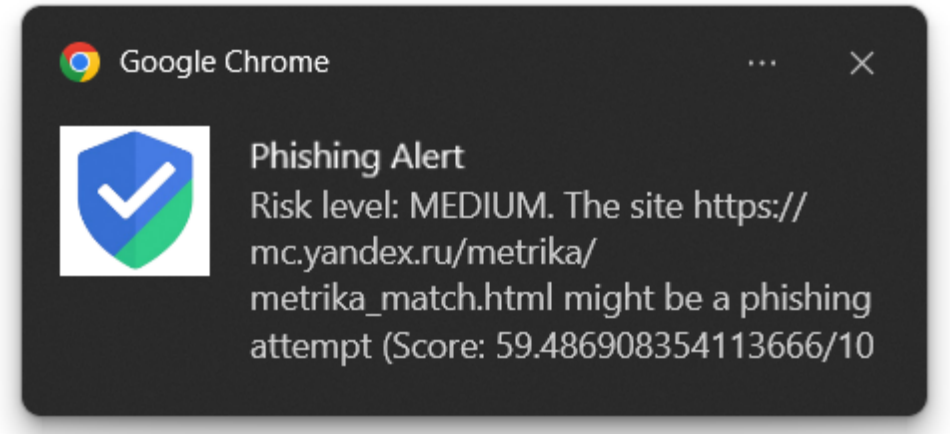


Figure 4.7: Content script reaction to dangerous page with medium or high risk

```
// Retrieve the current page URL
const currentUrl = window.location.href;

// Send the URL to the server for analysis
fetch('http://localhost:5000/predict', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ url: currentUrl })
})
.then(res => res.json())
.then(data => {
    // Check the risk level returned by the server
    const risk = data.risk_score;

    if (risk >= 70) {
        // High risk: block content and show a warning
        document.body.innerHTML = `
            <div style="background-color:red; color:#fff; padding:20px;">
                <h1>Warning! This site has been flagged as phishing.</h1>
                <p>Access to this resource is blocked for your safety (risk
                ↪ ${risk}%).</p>
            </div>`;
    } else if (risk >= 40) {
        // Medium risk: display a warning banner
        const banner = document.createElement('div');
        banner.textContent = `Warning: This site appears suspicious (risk
        ↪ ${risk}%)`;
        banner.style.cssText = "background-color: yellow; color: black;
        ↪ padding: 10px;";
        document.body.prepend(banner);
    }

    // Save the latest analysis result for access from the popup (e.g., via
    ↪ chrome.storage)
    chrome.storage.local.set({ lastAnalysis: data });
});
```

Figure 4.8: Fragment of `content.js`: Sending the Current URL for Analysis and Responding to a Dangerous Site

As shown in Figure 4.8, the content script, similar to the popup script, sends a POST request with the current URL and processes the JSON response from the server. Depending on the `risk_score`, actions are taken to inform the user or block the page. In cases of high risk, the entire content of the page is replaced with a warning (explaining the reason for the block and showing the risk percentage); for medium risk, only a yellow banner is added at the top of the page. For subsequent synchronization with the extension interface, the script saves the received analysis data (e.g., using `chrome.storage.local`)—this allows the popup script on the **Current Site** tab to immediately display the result to the user without making an additional request.

Manifest File (`manifest.json`)

The manifest is the configuration file of the Chrome extension that defines its core parameters, permissions, and component scripts. The developed extension uses the Manifest V3 format. Table 4.1 lists the key fields of the `manifest.json` file along with their values, followed by a corresponding JSON fragment (Figure 4.9).

Table 4.1: Main parameters of the extension manifest

Parameter	**Value**	**Description**
`manifest_version`	3	Format version of Manifest (Chrome MV3)
`name`	"Phishing Detector"	Extension name
`version`	"1.0"	Extension version
`action`	{ ¨ default_popup": ¨ popup.html" }	Call HTML-file popup interface
`content_scripts`	list with settings object	Identify connection with `content.js`
`permissions`	["storage", "activeTab"]	Permission: storage access, active tab access
`host_permissions`	["http://localhost:5000/*"]	Permission to access the local Flask server (CORS)

As shown, the `content_scripts` section configures the content script's behavior, specifying the URLs it should run on (`"matches": ["<all_urls>"]`, meaning all sites) and when it should be triggered (`"run_at": "document_start"`, i.e., as soon as the page begins to load).

The `permissions` field declares the extension's access rights: `storage` allows saving settings and history, and `activeTab` permits reading the currently active tab URL (although in this implementation, `content.js` directly handles that).

The `host_permissions` section includes the address of the local Flask server (`http://localhost:5000`http://localhost:5000), which is necessary to bypass Chrome's default cross-origin restrictions (CORS), allowing the extension's scripts to communicate directly with the backend API.

```
{
  "manifest_version": 3,
  "name": "Phishing Detector",
  "version": "1.0",
  "action": {
    "default_popup": "popup.html"
  },
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["content.js"],
      "run_at": "document_start"
    }
  ],
  "permissions": ["storage", "activeTab"],
  "host_permissions": ["http://localhost:5000/*"]
}
```

Figure 4.9: Fragment of `manifest.json` with core extension settings

This configuration enables the extension to operate efficiently and securely, integrating its client-side logic with the server-side phishing detection service via RESTful API calls.

System Component Interaction

Having described the client-side and server-side parts, let us now consider the workflow of how all system components interact. The system operates as follows:

1. **Page Load and Initialization:** When the user opens a webpage, the browser begins loading its content. Simultaneously, the content script is injected (at `document_start`, as defined in the manifest). It extracts the current URL and asynchronously sends it to the Flask server for analysis. Meanwhile, the page continues to load. If the server returns a high-risk result, the script can stop loading (e.g., using `window.stop()`) or replace the content.

2. **Server-side Analysis:** The Flask server receives a POST request at a dedicated API route (e.g., `/predict`). The server first checks whether the URL is whitelisted (such as system domains or user-approved entries from the Settings tab). Then it uses `utils.py` to extract dozens of structural and semantic URL features. Optionally, the server queries external services: Google Safe Browsing and VirusTotal. These provide additional indicators such as "known as malicious" flags. The resulting features are normalized and passed to the pre-trained machine learning model. The

model outputs a phishing probability (as a risk score in percentage). A structured JSON response is generated containing fields such as:

- `phishing`: boolean prediction result,
- `risk_score`: percentage score,
- `gsb_malicious` and `vt_malicious`: external API flags,
- `features`: raw feature data for debugging or display.

3. **Client-side Processing:** The content script receives the JSON response. For low risk, the page continues to load, and a "safe" status may be saved locally. For medium or high risk, a visual banner or full-page block is shown. When the user clicks the extension icon, the popup opens and displays the **Current Site** info. The popup script fetches the active tab's URL and looks for analysis results in local storage. If not found, it may request fresh data from the server. The **History** tab loads stored analysis logs from `chrome.storage`, and the **Check URL** tab allows manual queries. **Settings** lets users control GSB/VT usage, risk thresholds, clearing history, or maintaining a personal whitelist.

This setup offers layered protection—local ML-powered detection augmented by global threat intelligence. Components interact in real-time, balancing clarity (informative results and history) and transparency (non-intrusive behavior unless threats are found).

Server-side Implementation (Flask). The backend is built using Flask, enabling simple route definition and request handling. The core logic resides in app.py, which defines routes such as /predict (the main analysis endpoint). This route accepts POST requests with a URL and returns JSON-formatted analysis results. Below is a simplified route implementation:

```
{
  from flask import Flask, request, jsonify
import joblib, utils

app = Flask(__name__)
model = joblib.load('model_rf.pkl')  # Load the pre-trained model

# Example whitelist
WHITELIST = {"google.com", "wikipedia.org", "localhost"}

@app.route('/predict', methods=['POST'])
def predict_url():
    data = request.get_json(force=True)
    url = data.get('url')
    if not url:
        return jsonify({"error": "No URL provided"}), 400

    # Whitelist check
    domain = utils.get_domain(url)
    if domain in WHITELIST:
        return jsonify({
            "url": url,
            "phishing": False,
            "risk_score": 0,
            "message": "URL is whitelisted as safe"
        })

    # Feature extraction and prediction
    features = utils.extract_features(url)
    X = utils.prepare_features([features])
    proba = model.predict_proba(X)[0][1]
    risk_score = int(proba * 100)
    phishing = risk_score >= 50

    # External service checks
    gsb_flag = utils.check_safe_browsing(url)
    vt_flag = utils.check_virus_total(url)

    result = {
        "url": url,
        "phishing": phishing,
        "risk_score": risk_score,
        "gsb_malicious": gsb_flag,
        "vt_malicious": vt_flag,
        "features": features
    }
    return jsonify(result)
```

Figure 4.10: Fragment from `app.py`: Initialization of the Flask Application and Loading of the Pre-trained Machine Learning Model

This route handles core logic: retrieves the URL from the request, checks the whitelist, extracts features via `utils.py`, formats them for the model, and computes phishing probability. Additional threat checks (GSB, VT) are added to the response. The final result is returned as structured JSON.

The server may also define additional routes such as `/` (home), `/status` (to check service health), logging routines, or error handlers to ensure robustness and traceability.

Machine Learning Model (Training and Integration).

To classify URLs as phishing or benign, a machine learning model was developed and trained separately from the main system pipeline using the script `train_model.py`. The `phishing.csv` dataset served as the training data and included a wide range of labeled URLs—both phishing and legitimate.

The feature set was designed based on phishing detection research and URL analysis. It captured several aspects of a URL:

- **Structural anomalies:** presence of the `'@'` symbol (rare in normal URLs, may indicate redirection), hyphens in domain names (often used in deceptive subdomains), overly long URLs or domain names, and total character count.
- **Domain-based features:** use of an IP address instead of a domain name (often used in phishing), number of dots in the domain (excessive subdomains may mimic legitimate domains), unusual top-level domains (TLDs), encoding inconsistencies (e.g., `punycode`).
- **Content-based indicators:** suspicious keywords like "login," "secure," "verify," "account"; number of query parameters (use of `?` and `&` may imply malicious scripts); redirection patterns (extra `//` or embedded `http://`).
- **HTTPS and port indicators:** whether the URL uses SSL (starts with `https://`), use of non-standard ports (phishing sites may operate on unusual ports).
- **Other features:** domain association with authoritative entities (e.g., substring checks for impersonation), results from external services such as Safe Browsing and VirusTotal used as binary indicators.

Feature extraction and model training were handled in `train_model.py`. Several popular algorithms were tested, including Random Forest, SVM, Logistic Regression, k-Nearest Neighbors (k-NN), and XGBoost. After cross-validation and hold-out testing, Random Forest outperformed others, achieving accuracy exceeding 92% on the test data.

The final model used `RandomForestClassifier`, selected for its strong balance of precision and recall, and its robustness to diverse URL features. The trained model

was saved to disk using the `joblib` library and loaded at server startup in `app.py` (see Figure 4.10).

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from joblib import dump

# Load dataset with precomputed features
data = pd.read_csv('phishing.csv')
X = data.drop('label', axis=1)   # 'label' is the target variable (0/1)
y = data['label']

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
↪  random_state=42)

# Train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate accuracy on the test set
accuracy = model.score(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

# Save the trained model to file
dump(model, 'model_rf.pkl')
```

Figure 4.11: Fragment from `train_model.py`: Training a Random Forest Model on the URL Dataset

`utils.py` Module (Feature Processing Logic).

The `utils.py` module includes auxiliary functions for both server operations and model training. A core component is the `extract_features(url)` function, which computes the full set of URL features and returns them as a dictionary or ordered array. The module may also contain utilities for domain parsing, interaction with external APIs, and preprocessing.

The **main focus** is on parsing URLs and evaluating phishing-specific conditions as previously outlined.

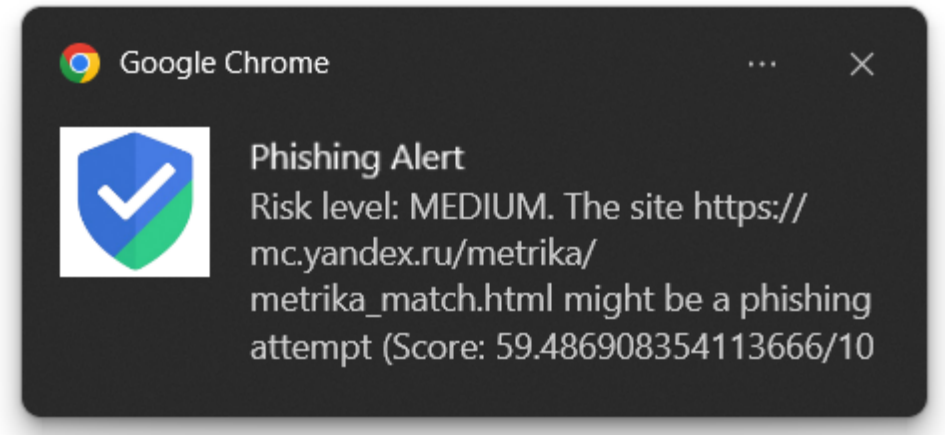


Figure 4.12: Fragment from `utils-parse`: illustrates a simplified implementation of the feature extraction logic

As shown in Figure 4.12, this simplified function uses the `urllib.parse` module to extract the domain name and regular expressions to check for patterns. It calculates numerical features (URL length, domain length, number of dots) and boolean features (presence of `https`, `@` symbol, hyphen in domain, use of IP address, suspicious keywords, and double slashes in the path).

In the full implementation, the feature set is broader (as described earlier), but the shown snippet reflects the general approach. Additionally, the `utils.py` module includes integration with external services: for example, `check_safe_browsing(url)` sends a request to Google Safe Browsing API with an access key and interprets the returned status (safe, suspicious, malicious) to return a Boolean. Similarly, `check_virus_total(url)` interacts with VirusTotal or uses preloaded data.

The module may also include a `prepare_features(features_dict)` function to format the features dictionary as a `numpy` array or `DataFrame` compatible with the model (e.g., ordering columns, normalizing values if required during training). As a result, the server provides detailed analysis output, while the extension transforms it into intuitive visual indicators. This separation of responsibilities (extension – interface and data capture, server – computational analysis) and modularity (utils, separate model training process) makes the system flexible and scalable for future enhancements.

4.1.4 Justification of Tools and Models

Choice of Server Framework (Flask vs Django)

For implementing the server-side component of this project, the Flask framework was selected. The primary reason is that Flask is a lightweight microframework ideal for creating REST API services with minimal boilerplate code. Since the server's functionality is limited to handling a few HTTP requests (mainly a single endpoint for URL analysis) and does not require a complex web interface, databases, user authentication, or other "heavy" features, using a full-fledged framework like Django would be excessive.

Flask allows rapid deployment of an API server: the developer defines the routes and logic manually without adhering to a rigid project structure. As a result, the Flask application file (see Figure 4.10) remains compact and readable, simplifying debugging and maintenance. Moreover, Flask offers high customization capabilities: it is easy to integrate additional libraries (e.g., for CORS when hosted separately) and configure custom error handling.

In contrast, Django is a "complete" framework designed for large web applications, with built-in components like ORM, templating engine, authorization system, and admin panel. For the task of phishing URL detection, such functionality was unnecessary. Django would add significant overhead in terms of loading, configuration, and resource consumption. Flask turned out to be more suitable for fast prototyping and deployment of the URL analysis service.

Importantly, Flask also simplifies integration with machine learning: the model is loaded directly into memory, and requests are processed on the fly. Django, with its more complex request lifecycle, would require additional effort for such optimizations. Therefore, Flask was chosen based on the principle of sufficiency and simplicity: it provides everything necessary to implement the API and nothing superfluous, aligning with the microservice architecture used in this project.

Choice of Machine Learning Algorithm (RandomForestClassifier).

During the development of the system, several classification algorithms were evaluated for the task of binary URL classification (phishing vs. benign). The candidates included Support Vector Machines (SVM), Logistic Regression, k-Nearest Neighbors (KNN), Gradient Boosting (specifically `XGBoost`), and Random Forest.

Random Forest was ultimately selected for several reasons. First, as shown in the experiments (see Table 4.2), the Random Forest model demonstrated the highest accuracy on the test data ($> 92\%$), outperforming close competitors. Second, Random Forest is robust to overfitting and works well with diverse features, including numerical, categorical, and boolean values. The project's feature set includes a combination of string lengths, character counters, and logical flags.

The bagging approach underlying Random Forest makes it less sensitive to noise in individual features: even if some URLs contain anomalous values, the ensemble of decision trees reaches a reliable conclusion by averaging across multiple features. Third, Random Forest enables feature importance estimation—an asset for analytics, allowing identification of which URL characteristics most contribute to phishing detection. SVM and Logistic Regression are less interpretable in this context (especially SVM with nonlinear kernels). Gradient Boosting (XGBoost) can also achieve high accuracy but is more complex to tune and requires more resources during training. In this case, the accuracy difference between XGBoost and Random Forest was minor, leading to a preference for the simpler-to-use Random Forest. Lastly, prediction speed was a critical factor: RandomForestClassifier from scikit-learn is very fast at inference (tens of shallow trees are evaluated in milliseconds), meeting the real-time requirements of a browser extension. Thus, the choice of Random Forest is justified both by empirical results and theoretical suitability for the task.

Feature Selection for URL Analysis

As outlined in Section 3.1, the system employs a set of features characteristic of phishing URLs. The choice of these features is based on an analysis of typical differences between malicious and legitimate links. Cybersecurity literature and practice describe several heuristic rules for phishing detection: for instance, very long or obfuscated URLs often indicate suspicious behavior; the `'@'` symbol rarely appears in normal URLs and can be used by attackers to mask true redirection targets; using an IP address instead of a domain is a red flag, as legitimate brands generally avoid this. Features like the presence of words such as `"login"`, `"secure"`, and `"verify"` were also selected—phishing pages often include such keywords in their URLs to gain user trust (e.g., `secure-verification-paypal.com`), which genuine sites like PayPal do not.

These heuristics are statistically supported: the dataset used for model training showed that phishing URLs frequently contain such patterns compared to regular sites.

Therefore, the selected features are informative—each contributes to threat detection. Moreover, these features are simple and fast to compute: they are derived via URL string parsing and basic operations like regex matching or character counting, which require minimal processing power and can be executed in real time. This is critical for client-side usage: heavyweight operations (e.g., page content loading or JavaScript execution) are avoided, reducing latency.

Another benefit is generalizability: the features are domain- and language-independent and applicable to URLs globally. As a result, the trained model can effectively detect previously unseen phishing links by relying on general probabilistic trends. Lastly, the feature set is extensible—the modular `utils.py` structure allows new checks to be added (e.g., OCR-based page text analysis, domain age verification via WHOIS). For this reason, the system combines rule-based feature extraction with model-based prediction in a flexible and efficient manner.

Comparison of Machine Learning Models

Throughout development, a comparative analysis was conducted to evaluate several classification models. Each model's performance was assessed using test set metrics: overall accuracy, precision, recall (phishing detection), and F1-score. The averaged results are presented in Table 4.2.

Table 4.2: Comparison of Classification Algorithms by Prediction Quality

Model	**Accuracy, %**	**Precision, %**	**Recall, %**	**F1-score, %**
Random Forest	92.4	93.0	92.0	92.5
SVM (Support Vectors)	90.5	89.7	91.8	90.7
k-NN (k=5)	85.0	82.1	87.5	84.7
XGBoost (depth=3)	91.7	92.5	90.2	91.3

As shown in Table 4.2, the Random Forest model demonstrated the highest accuracy (approximately 92.4%) along with balanced precision and recall values (˜93% and ˜92%, respectively). This indicates that the model is capable of detecting the majority of phishing URLs (high recall) while maintaining a relatively low number of false positives (high precision).

The `XGBoost` model came close in performance (about 91.7% accuracy), slightly behind Random Forest. Its precision was even slightly higher, but recall was slightly

lower, suggesting a possible tendency to miss some phishing sites. Nevertheless, the difference between these two ensemble methods was minor.

The SVM algorithm achieved around 90.5% accuracy. While it also performed well, its lower precision (approximately 89.7%) meant it produced more false alarms compared to Random Forest. This may be due to the nonlinear distribution of features, making it harder to tune the SVM kernel optimally without extensive parameter selection.

The k-Nearest Neighbors (k-NN) algorithm performed significantly worse than the others, with an accuracy of about 85% and particularly low precision (82.1%). In the context of phishing detection, this level of false positives is considered unacceptable. k-NN does not scale well with high-dimensional data and is prone to overfitting on local patterns, making it less suitable for this application.

Based on these results, Random Forest was selected as the optimal model. It provides high classification performance, is relatively easy to implement, and does not require extensive hyperparameter tuning (in the experiment, 100 trees with a depth of ∼5–6 were sufficient). The observed performance also outperforms SVM and especially k-NN in prediction speed, which is important for integration into a real-time browser extension.

Overview of Used Libraries

The following libraries and tools were used in the project:

- **Flask 2.x** – for implementing the server-side web application. Flask provided essential classes and functions to run the web server, route decorators (e.g., `@app.route`), the request object for accessing request data, and `jsonify` for forming JSON responses. Flask simplified the creation of the REST API, allowing focus on data processing logic.
- **scikit-learn 1.x** – as the main machine learning framework. Key components used include model classes like `RandomForestClassifier`, `SVC` (for SVM),

  `KNeighborsClassifier`, and `LogisticRegression`, as well as utilities such as `train_test_split`, `accuracy_score`, `precision_score`, and `recall_score`. The unified fit/predict interface made it easy to compare algorithms. The library was also used at the prediction stage — the trained Random Forest model was a scikit-learn object loaded into the Flask app for URL classification.

- **joblib** – a serialization utility from the scikit-learn ecosystem, used to save the trained `RandomForestClassifier` model as `model.pkl` and load it on the server.
- **Pandas** – used for saving large objects like NumPy arrays, which are typical in training model files, offering better performance and smaller file size than standard `pickle`.
- **Pandas (again)** – used during dataset preparation and analysis. In the training script (`train_model.py`), `pandas` was used to load CSV files, preprocess data (e.g., feature generation using `apply` functions), and possibly generate reports on feature distributions.
- **Requests (or a similar HTTP library)** – used in `utils.py` to interact with external APIs like Google Safe Browsing and VirusTotal. For example, `requests.post()` with JSON headers was used to send API requests, and the responses were parsed using `.json()` or other parsing techniques.
- **Chrome Extension API** – a set of built-in APIs in Chrome used in the frontend. While not a traditional library, it provided browser-level functionality for extension scripts. APIs such as `chrome.storage` (for saving history and settings), `chrome.tabs` and `chrome.runtime` (for accessing tab info and messaging between background, content, and popup scripts) were used. Additionally, `chrome.scripting` (in Manifest V3) was employed when needed for dynamic script injection or content removal.
- **XGBoost Library** – used during experimentation to train a gradient boosting model. It was implemented using `xgboost.XGBClassifier` and required installing the corresponding package. While it was not selected as the final model, it played a key role in comparative evaluation.
- **Additional tools** – standard Python libraries such as `re` (for regular expressions), `math`, `os`, and `logging` (for server-side logging) were also utilized. Google Safe Browsing and `VirusTotal` APIs were integrated using their standard REST interfaces via basic HTTP requests, without the need for specialized SDKs.

The choice of these technologies was based on their widespread adoption and usability. Scikit-learn and Pandas are industry standards for data analysis, Flask is one of the most popular lightweight web frameworks, and the Chrome Extension API is essential

for browser extension development. This ensures strong community support and ease of future maintenance, enabling other developers or researchers to understand and enhance the system using familiar tools.

4.2 Security and Data Protection

4.2.1 Security and Data Protection

In a phishing-detection browser extension, preserving user privacy and securing data are paramount. The system is designed so that user browsing behavior remains anonymous: no identifying information or personal data is collected beyond what is strictly necessary for functionality. In practice, this means that all URL and page analysis occurs locally, on the client side, rather than sending raw browsing data to an external server. For example, recent research on phishing extensions advocates entirely client-side models precisely to "enhance user privacy by keeping browsing data local" [7]. By processing suspected URLs on the user's device, the extension avoids transmitting any specific visited URL or user credentials, so there is nothing personally identifiable to log. This design directly addresses known privacy risks: as Olejnik et al. note, even subsets of browsing history can uniquely fingerprint and re-identify users [63]. Consequently, our system never logs or stores full URL strings or browser histories. Any telemetry or event logging is strictly limited to aggregated, non-identifying metrics (for instance, counts of detected phishing events) and is further de-identified. In line with OWASP guidance on logging, sensitive data and direct identifiers are omitted or pseudonymized wherever possible [66] [67]. In short, the extension's data-handling policy embodies privacy-by-design: user browsing data remains on-device and any minimal telemetry is anonymized so that no individual user can be reconstructed from the logs.

4.2.1.1 Security Mechanisms

To protect data in transit and prevent tampering, the extension enforces HTTPS (TLS) for all network communications. The remote server and any APIs it uses require TLS 1.2/1.3 with valid certificates, following industry best practices. Specifically, the server is configured to use modern cipher suites and HTTP Strict Transport Security (HSTS) so that browsers refuse any insecure (HTTP) fallback [**?**]. All extensions calls use HTTPS URLs, and certificate validation is enforced. This ensures that data (for example, any model queries or static file downloads) is encrypted end-to-end and cannot be intercepted

or modified by man-in-the-middle attackers. OWASP's User Privacy Protection guidelines stress that "user communications must be encrypted in transit" using "verified certificates [and] strong ciphers" [66]; our configuration adheres to this standard. In addition, the server sets cache-control headers (e.g. Cache-Control: no-store) so that no sensitive content is stored in browser or proxy caches. Likewise, any sensitive information (such as tokens or classification results) is sent in the message body rather than in URL query strings, preventing accidental leakage via browser history or referer headers. These measures follow the recommendation to "keep sensitive data out of the URL and cache". Together with strict TLS usage and HSTS, this prevents eavesdropping and downgrade attacks on the extension's network traffic.

Cross-Origin Resource Sharing (CORS) is carefully configured to allow only legitimate requests from the extension. Because extensions have unique origin identifiers (e.g. chrome-extension://<ID>), the server's Access-Control-Allow-Origin header is set to exactly that origin (or a small whitelist) rather than using a wildcard. This follows MDN's advice to specify "the minimum possible number of origins" in Access-Control-Allow-Origin, since otherwise "unauthorized origins [could] read the contents of any page on your site". In practice, the extension's fetch calls include only allowed origins, and the server refuses requests from other domains. Credentials (cookies or auth tokens) are only sent when strictly needed, and CORS's Access-Control-Allow-Credentials is not enabled for wildcard origins. By securing CORS in this way, we prevent malicious webpages from making unauthorized, credentialed requests to our backend, avoiding CSRF-like attacks.

Importantly, the design never stores or logs user URLs at rest. No browsing history or full website addresses are written to disk by the extension or server. This is a deliberate, explicit choice: as research shows, browsing histories are highly identifying [63]. Therefore, our system treats each URL analysis as ephemeral; once a site has been checked, its address is discarded. Any internal logging (for debugging or metrics) omits direct URL values entirely. In effect, the extension follows OWASP's principles to exclude personal data from logs and to minimize stored identifiers [67]. Even cached network responses contain only generic payloads (e.g. model weights), with no user-specific tokens. The extension's manifest and permission set also follow the least-privilege principle: it requests only the minimal permissions required (for example, access to page content on a limited set of domains rather than blanket "<all_urls>" permission). This limits attack surface and fits OWASP's guidance to "prioritize least privilege" in extension design [**?**]. In summary, by combining strict HTTPS, tight CORS, minimal permissions, and a policy

of not logging URLs, the system enforces multiple layers of defense around sensitive data.

4.2.1.2 Threat Mitigation

The extension's architecture proactively mitigates common web security threats. First, code injection and cross-site scripting (XSS) risks are countered by strict input/output controls and Content Security Policy (CSP). All inputs derived from webpages or external sources are sanitized or validated before use. For instance, when content scripts parse a page's DOM to check for phishing indicators, they do not execute or inject any untrusted scripts. The extension's manifest (using Manifest V3) includes a default CSP of default-src 'self', which blocks all inline scripts and remote code by default [**?**]. Only explicitly allowed scripts (the extension's own code) can run. This CSP enforcement is a key defense: OWASP emphasizes that without a strict CSP, "attackers can inject scripts" leading to XSS [67]. By disallowing eval() and any dynamic script insertion, and by encoding or stripping user-provided HTML, the extension eliminates XSS attack vectors. In addition, any output inserted into page overlays (such as warning banners) is done using safe DOM methods (e.g. textContent or vetted template literals) rather than raw HTML. These practices align with OWASP advice to "follow secure coding practices, enforce proper input validation and output encoding" to mitigate XSS.

Injection attacks (such as SQL or command injection) are likewise guarded against. Although the extension itself does not use a traditional database, its server-side component (if any) uses parameterized queries or ORM layers so that untrusted input cannot alter query logic. All parameters sent to the backend (e.g. extracted features from a URL) are treated as data only and are properly escaped. We adopt OWASP's core principle of never executing untrusted input as code. Furthermore, any dynamic data used in logging or diagnostics is sanitized to prevent "log injection" (e.g. stripping control characters) as recommended in OWASP's Logging guidance.

The system also prevents data leakage in other ways. As mentioned, no sensitive information is included in URLs or browser history. Any cookies set (such as a session cookie) are marked HttpOnly and Secure, so they cannot be read by scripts on a compromised page. We also set strict Referrer-Policy and Content-Security-Policy headers on pages served by our server, ensuring that no referrer data or unexpected resources leak across contexts. HTTP responses include X-Content-Type-Options: nosniff and disable caching of sensitive endpoints. These controls prevent attackers from exploiting cached data or intercepting leakage.

Additionally, the extension does not use external scripts or libraries from untrusted sources; all dependencies are locked to specific versions. The codebase is subject to regular security audits: third-party libraries and tools are kept up to date using automated scanning (e.g. npm audit or OWASP Dependency-Check) [67]. This guards against vulnerabilities in outdated components, aligning with OWASP's call to "regularly audit third-party dependencies". Finally, error handling and monitoring are designed to avoid exposure. The system logs only high-level events (e.g. "phishing site detected") without any PII. Any error messages returned to the user or logged on the server omit internal details and do not reveal stack traces. By combining input validation, secure libraries, strict CSP, encrypted transport, and logging discipline, the extension architecture mitigates injection/XSS and data-leakage threats comprehensively.

4.3 Testing

4.3.1 Testing Methodology

The testing of the developed system was conducted in multiple stages, combining both manual and automated approaches. In the first stage, the focus was on manual testing of UX and functionality: a group of expert users installed the extension in their browsers and tested it in real-world scenarios. They manually entered various URLs (both clearly safe and potentially phishing) through the extension interface and followed different links, observing the system's behavior.

The evaluation criteria included: stability of the extension (whether it caused any errors or browser crashes), correctness of the system's response to different inputs (e.g., URLs with or without the `http://` prefix), and clarity of the visual display (whether the color indicators and messages were understandable). Particular attention was paid to response time: the system had to analyze a link and return a result almost instantly (within one second), to avoid noticeable delay for the user.

In the second stage, semi-automated testing was conducted using precompiled URL datasets. Two datasets were used in the project: in addition to the training set `phishing.csv`, an independent set `malicious_phish.csv` was used for testing. This dataset contained links from various sources (phishing campaigns, safe site collections, random new addresses). A script was written to programmatically send requests to the local Flask server using URLs from this list and compare the results with known labels

(i.e., whether a URL was phishing or not).

This approach allowed automatic collection of model performance statistics and identification of potential gaps in rule coverage. Edge cases were also tested, such as:

- URLs of local resources (e.g., `http://localhost`, `http://127.0.0.1:port`), which should not be misclassified as phishing;
- Very long URLs (hundreds of characters, often used for spam redirects), which the system must be able to process;
- URLs with unusual but legitimate structures (e.g., URLs with parameters, hashes, internationalized domain names).

These tests confirmed the system's robustness across different URL formats.

Test Cases and Observations

Specific scenarios were evaluated during manual testing to illustrate how the system behaves:

- **Safe URLs of well-known sites.** For example, when visiting `https://www.youtube.com`, the extension displayed a "Safe" status with a low risk score (below 20). As mentioned in Figure 3.2, a green notification was shown for YouTube. In this case, no suspicious features were detected: the domain is trusted, the URL contains no anomalies, and the model correctly classified it as safe. Opening the popup showed a green indicator and a safety message.
- **Suspicious and local addresses.** URLs such as `http://127.0.0.1:8080/test` (a local server) and non-existent domains like `http://example.local/login` were tested. The extension marked them as "suspicious," with medium risk scores (e.g., between 40–60 out of 100). This was expected: using an IP address as a domain increased the risk score (triggering the `has_ip` feature), but these links were not in phishing databases, so the model did not assign maximum risk. In these cases, the interface displayed a yellow background with a warning, but did not fully block the site. The user was informed that the address was unusual, but the final decision (to proceed or leave the site) remained with them.

- **Known phishing URLs.** An example is a link mimicking a popular service: `http://paypal-login.verification.ru/...`. This URL exhibits clear signs of a phishing attack: the domain contains the words "paypal" and "login," the domain name is long, and the keyword "verification" is present. The model assigned it a high risk score (over 80 out of 100). The content script immediately blocked the page load and displayed a red warning (Figure 3.3 shows a similar block event). In the popup interface, the site was marked as "Phishing detected" with a red icon. Such links are often also detected by Google Safe Browsing—in tests, if the URL was listed in the GSB database, the response field `gsb_malicious=true` confirmed the threat.

Results and Accuracy Metrics

To quantitatively assess the system's effectiveness, key classification metrics were calculated using a test set of 500 URLs (not used during training). The results were as follows:

- Total number of URLs tested: 500
- Correctly classified (True Positives + True Negatives): 462
- False Positives: 18
- False Negatives: 20
- Accuracy: 92.4%

These figures were obtained using an automated script and confirmed the high effectiveness of the model and overall system reliability. For clarity, the classification confusion matrix is presented in Table 4.3.

Table 4.3: Confusion Matrix of the Model on the Test Set (N = 500)

Predicted class	**Secure site**	**Phishing site**
Secure site (legit)	TN = 232	FP = 18
Phishing (attack)	FN = 20	TP = 230

Out of **250** known safe links, 232 were correctly identified as safe (true negatives), while 18 were incorrectly flagged as phishing (false positives). Of the 250 phishing URLs,

230 were correctly classified as phishing (true positives), and only 20 were missed by the model (false negatives, i.e., undetected attacks).

Based on this data, the system achieved an accuracy of **92.4%** (as previously noted), a phishing detection precision of approximately **92.7%**, and a recall of approximately **92.0%**. In other words, about **93%** of the links marked by the system as phishing were indeed phishing, and the system detected approximately **92%** of all phishing threats presented in the test set.

These results indicate that the selected Random Forest model and the combination of features effectively handle the task: the rate of false alarms and missed attacks remains low.

The graphical representation of the system's performance during testing is shown in Figure 3.4 (a screenshot of the *History* tab of the extension after multiple checks): green markers indicate safe URLs, yellow markers indicate suspicious ones, and red markers represent blocked phishing sites. This visualization provides a convenient overview of how many threats were prevented and how many links were safe.

User Interface Examples and Experience

The extension's interface proved intuitive even for untrained users during testing. Figure 3.2 displayed the popup window result for the current site: a **large color-coded** indicator and an explanatory label made it easy to determine the status at a glance. Users appreciated the availability of detailed results—if desired, they could open the *History* tab to view the risk levels numerically and review which sites were analyzed. Figure 3.4 illustrates this functionality: the log contains entries such as "URL – date/time – result (Safe/Phishing) – risk %." This enables users to analyze their browsing history and how the system responded.

During UX testing, it was found that automatic blocking (as shown in Fig. 3.3), while protective, could be inconvenient in borderline cases. As a result, in the *Settings* tab (Fig. 3.5), the default behavior is set to show warnings for medium risk, while full blocking for high risk can be optionally disabled by the user. Overall, the test group found the extension's functionality sufficient and helpful: the system clearly highlights potentially dangerous links without overwhelming the user with excessive information. Performance was also praised—interface responses were smooth, and server requests were

processed quickly (on average, ∼0.2–0.3 seconds per URL analysis), making the process seamless during regular browsing.

Conclusions and Recommendations for Improvement

Based on the test results, the developed system effectively detects phishing URLs and serves as a reliable tool for protecting users against phishing. The machine learning model demonstrated high accuracy, and integration with external services increased trust in the analysis results. Nevertheless, several areas for future improvement were identified:

- **Expand the training dataset** to several million entries by incorporating fresh data on phishing attacks (e.g., via regular use of APIs like *PhishTank*). This would allow retraining the model on a more representative sample and improve its ability to detect emerging threats.
- **Implement page content analysis** in addition to URL analysis—this includes analyzing the HTML code and scripts of loaded pages. For example, identifying phishing characteristics in the page text (e.g., password forms on unsecured connections, suspicious JavaScript functions). This would create a hybrid system (URL + content) and make it harder to bypass protection through simple link modifications.
- **Introduce an image analysis module (OCR)**: Phishing sites may display text (e.g., bank logos, instructions) as images to bypass text filters. Embedding OCR to recognize text from screenshots would help detect brand mentions or warnings directly within images.
- **Support a user feedback mechanism**: Allow users to report missed phishing sites (if any) or false positives (if a safe site is blocked). This data could be sent to developers or used for continuous model retraining to enhance quality over time.
- **Increase cross-platform support and usability**: Adapt the extension for other browsers (Firefox, Safari) and potentially for mobile platforms. Also, consider implementing a standalone application or integrating with email clients to check links in incoming messages—thus expanding the use cases of the detection algorithms.

In conclusion, the diploma project achieved its goal by creating a working system for detecting phishing URLs and demonstrating its effectiveness. The final recommendations

highlight directions for further development to ensure the system remains relevant and robust against evolving cybersecurity threats.

4.3.2 Minimizing Security Risks

Several common vulnerabilities were addressed during development to ensure the reliability of the system:

- **Input Validation:** All user-submitted data is validated and sanitized to prevent code injection or malformed data attacks.
- **Model Safety:** The machine learning model is static and maintained offline, preventing any dynamic retraining that could be exploited.
- **Safe Interface Design:** The extension's interface does not display or render untrusted content, minimizing the risk of front-end vulnerabilities such as cross-site scripting.

4.3.3 Alignment with Security Standards

The security practices followed during development are consistent with widely accepted industry standards. Key areas of compliance include secure communication, minimal data collection, proper input handling, and controlled access policies. These practices help ensure the extension is secure, responsible, and respectful of user privacy.

4.3.4 Results

To verify the effectiveness of the security measures, a series of tests were conducted, including both manual review and automated vulnerability scanning. The results are summarized as follows:

- No sensitive data was stored or exposed during any test session.
- All network requests were properly encrypted and limited to authorized domains.
- Attempts to inject malicious URLs or scripts were successfully blocked by input validation mechanisms.

- The extension interface remained stable and secure during use, with no cross-site scripting (XSS) or code execution issues detected.

Chapter 5

CONCLUSION

The methodology outlined in this chapter describes a comprehensive and layered approach to phishing detection, integrating supervised machine learning, in-browser structural analysis, and external reputation services. Each component was selected based on established practices in the cybersecurity domain and validated through current academic literature.

The use of the Random Forest algorithm enables accurate binary classification while offering interpretability through feature importance scores. The incorporation of Google Safe Browsing and VirusTotal APIs adds redundancy by cross-referencing predictions with authoritative external databases, thereby enhancing overall reliability. In parallel, the DOM analysis module strengthens detection capabilities by identifying suspicious elements embedded in the structure and behavior of web pages—particularly useful for recognizing zero-day threats and impersonation attempts.

The system's modular architecture allows for individual components to be updated or extended independently, supporting long-term adaptability. Communication between components is handled through a RESTful API interface, promoting scalability and ease of integration with additional services or models in the future.

By implementing this detection pipeline within a Google Chrome extension, the solution delivers real-time, context-aware protection directly in the user's browsing environment.

Annex1

Appendix A: Overview of Safe Click – Phishing Prevention

Safe Click is a browser extension designed to detect and prevent phishing attacks in real time. It combines supervised machine learning, dynamic feature extraction, real-time DOM analysis, and integration with external threat intelligence APIs (Google Safe Browsing, VirusTotal) to provide robust protection for end users.

Key features of Safe Click include:

- Real-time analysis of URLs using a trained machine learning model.
- Extraction of 34 structural, lexical, and semantic features from each URL.
- Real-time DOM analysis to detect suspicious forms, hidden elements, and phishing-related keywords within the page content.
- Integration with Google Safe Browsing and VirusTotal APIs for additional threat intelligence.
- User-friendly interface for risk notifications, history, and advanced settings.
- Customizable risk thresholds and notification preferences.

Appendix B: Installing and Configuring Safe Click

This section provides a step-by-step guide for installing and configuring the Safe Click browser extension and its supporting machine learning server. The procedure ensures reproducibility and facilitates independent verification of the system's functionality.

1. Downloading the Extension

Obtain the extension package from this link (URL) provided by the project supervisor or repository.

2. Installing Python

Download and install Python version 3.11 from the official website 1. Note: During installation, ensure that the option "Add Python to PATH" is selected to enable command-line access to Python.

3. Installing Project Dependencies

Open a terminal (for example, via Visual Studio Code: View Terminal) and navigate to the root directory of the project. Install all required Python dependencies by executing:

- pip install -r requirements.txt

4. Launching the Machine Learning Server

Start the machine learning server, which is required for real-time URL analysis, by running the following commands in the terminal:

- cd ML_Server
- cd server
- python app.py

Important: The server process must remain active while the extension is in use, as it handles all model inference requests.

5. Loading the Chrome Extension

Open the Google Chrome browser and navigate to chrome://extensions/. Enable "Developer mode" using the toggle in the upper right corner. Click "Load unpacked" and select the phishing_extension_improved directory from your local project files.

6. Verifying Installation

Visit any website in Chrome. Confirm that the Safe Click icon appears in the browser toolbar. The extension should automatically analyze the current page and display a security assessment.

Troubleshooting

Server connection issues: Ensure that Python 3.11 is installed and all dependencies are present. The ML server must be running (see Step 4). Extension not functioning: Verify that the correct project folder was selected and that the server is active.

Appendix C: Examples of Using Safe Click

Example 1: Real-Time Phishing Detection

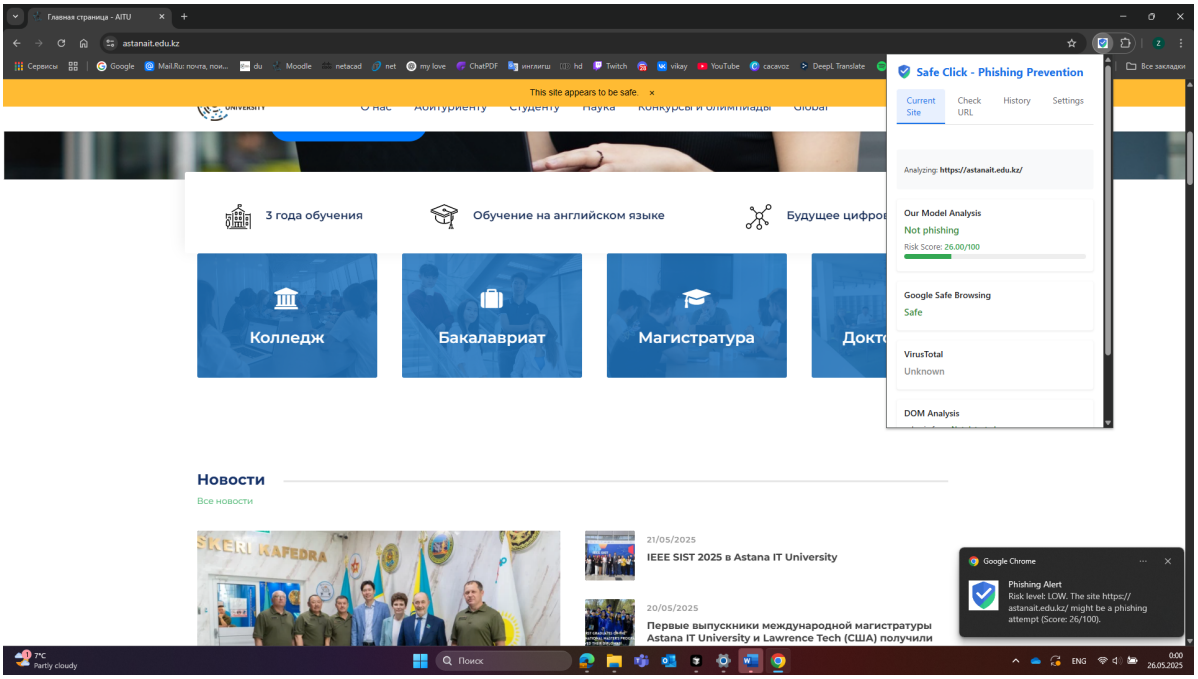
Scenario: A user navigates to a potentially suspicious website in Google Chrome.

Process:

- The Safe Click extension automatically extracts the URL of the current page.
- The URL is sent to the local machine learning server, where 34 features are dynamically extracted and analyzed by the trained Random Forest model.
- In addition to URL analysis, the extension performs real-time DOM analysis on the loaded web page. The DOM analysis module checks for:
  - Presence of login forms or password fields
  - Suspicious keywords in the visible text (e.g., “verify your account”, “update your information”)
  - Abnormal or empty form actions (e.g., forms submitting to external domains)
  - Hidden iframes or external links in forms
  - UI restrictions such as disabled right-click
- The results of the DOM analysis are combined with the machine learning model’s prediction and, if available, external threat intelligence from APIs.

- The extension displays a risk assessment banner at the top of the page:
  - Red banner: High risk (phishing likely)
  - Yellow banner: Low risk (site appears safe)
- Simultaneously, a browser notification is generated, summarizing the risk level and providing additional details.

Outcome: The user receives immediate, actionable feedback regarding the safety of the visited website, enabling informed browsing decisions.



Current Site" tab of the Safe Click extension: analysis results of the web page

Example 2: Manual URL Checking ("Check URL" Feature)

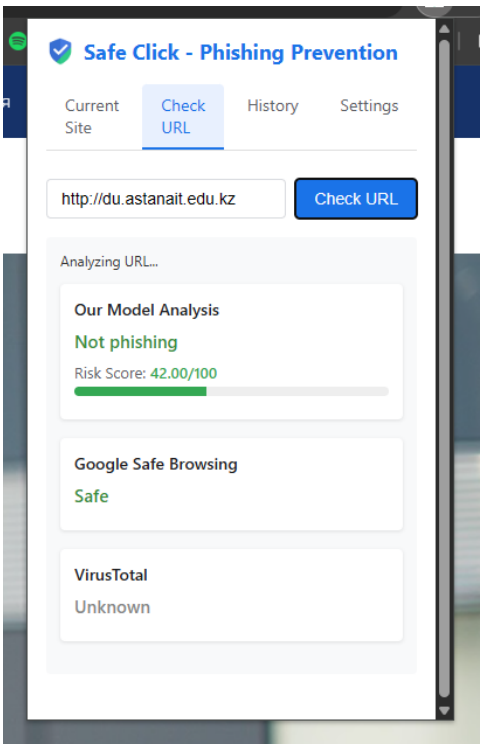
Scenario: A user wants to manually check the safety of a specific website before visiting it.

Process:

- The user clicks the Safe Click extension icon in the browser toolbar to open the popup window.

- In the “Check URL” tab, the user enters the desired website address (URL) into the input field.
- The user clicks the “Check URL” button.
- The extension sends the entered URL to the machine learning server, where features are extracted and the risk is assessed.
- The result is displayed directly in the popup, showing:
  - The computed risk score
  - A verdict (e.g., “Safe”, “Suspicious”, or “Phishing”)
  - Additional details from external threat intelligence APIs, if available

Outcome:The user receives an instant security assessment of any URL, allowing them to make informed decisions before navigating to potentially dangerous sites.



"Check URL" tab of the Safe Click extension: analysis results of the URL

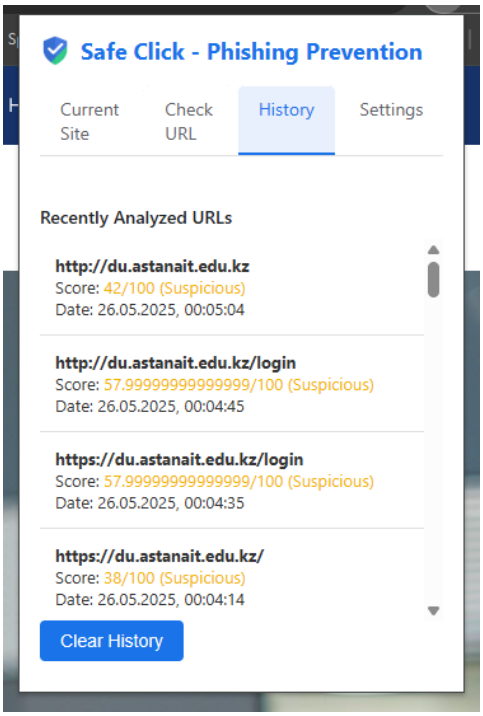
Example 3: Reviewing Analysis History

Scenario: A user wants to review previously analyzed URLs and their risk assessments.

Process:

- The user opens the extension popup and selects the "History" tab.
- A chronological list of recently visited URLs is displayed, including:
  - The analyzed URL
  - The computed risk score
  - The final verdict (safe, suspicious, or phishing)
  - The date and time of analysis

Outcome: The user can audit their browsing history for potential threats and gain insights into the extension's decision-making process.



"History" tab of the Safe Click extension

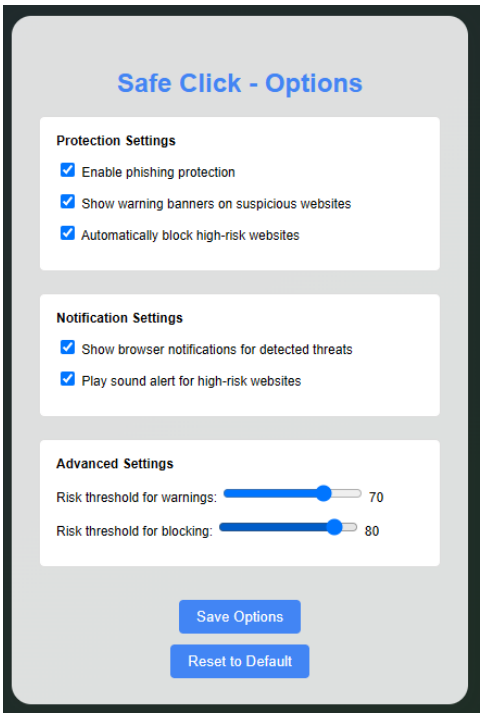
Example 4: Adjusting Risk Thresholds

Scenario: A user wishes to customize the sensitivity of phishing detection.

Process:

- The user opens the Safe Click extension popup and navigates to the “Settings” or “Options” page.
- Using the provided sliders, the user sets custom values for:
  - Risk threshold for warnings (controls when banners are shown)
  - Risk threshold for blocking (controls when access is automatically blocked)
- The extension applies these thresholds in real time, allowing the user to balance security and usability according to personal preferences.

Outcome: The extension’s behavior dynamically adapts to user-defined risk tolerance, supporting both conservative and permissive browsing styles.



"Settings" of the Safe Click extension

Appendix E: Practical Application of Safe Click in Cybersecurity

- Provides real-time, automated protection against phishing attacks by analyzing URLs and web page content before users interact with them.
- Enhances defense in depth by combining machine learning, DOM analysis, and external threat intelligence (Google Safe Browsing, VirusTotal).
- Supports user security awareness through clear risk notifications and customizable protection settings.

Annex 2

Appendix A: Survey Overview

A usability survey was conducted among **30 participants**, primarily in the **18–34 age group**, to evaluate the effectiveness, usability, and feature set of the browser extension *Safe Click – Phishing Prevention*. Participants watched a demonstration video before answering. The survey questionnaire is available online at Google Form, and the summary of responses is provided in the Excel report.

Appendix B: Key Statistics

Summary of User Survey Results

Metric	**Result**
Average Rating (1–10)	9.2
% of Users Who Found Interface "Very Easy to Understand"	76.7%
% Who Rated Security Checks as "Extremely Useful"	63.3%
Most Expected Accuracy	80–89% (53.3%), 90–100% (40.0%)
Top Features Identified as Valuable	Real-time analysis, Multi-check
% Who Value Custom Risk Thresholds as Important (rated 4 or 5)	90%
Top Notification Method	Warning banners + Auto-blocking
% Who Find History Feature "Very Valuable"	73.3%
Top Requested Features	Versions for other browsers, explanation of risk score, onboarding hints

Appendix C: Most Valued Features (Top 3)

From multiple-choice inputs:

- **Real-time Website Analysis** – mentioned by **80%**
- **Multiple Security Sources (GSB, VirusTotal)** – **76.7%**
- **DOM Analysis (e.g., login forms)** – **50%**
- **Risk Score Display** – **46.7%**
- **Customizable Risk Thresholds** – **43.3%**
- **History Feature** – **30%**

Appendix D: Most Effective Notification Types

(Selected all that apply)

- Warning Banners on Websites – **90%**
- Auto Blocking of High-Risk Sites – **86.7%**
- Browser Notifications – **63.3%**
- Sound Alerts – **36.7%**

Appendix E: Qualitative Feedback Highlights

Participants praised:

- Clear interface and numeric risk scoring
- Transparency via VirusTotal + Google Safe Browsing results
- The ability to **customize warning/blocking thresholds**

Suggested improvements:

- Add **educational explanations** for why URLs are risky
- Include **color-coded risk score** (e.g., red/yellow/green)
- Provide a **"Report this site"** button for users
- **Onboarding tutorial** and **cross-device sync**
- Make extension available in **other browsers (Firefox, Safari)**

Appendix F: Conclusion

The survey confirms that Safe Click is perceived as:

- **Highly usable** and **visually intuitive**
- Trusted due to **multi-source verification**
- Ready for real-world deployment, especially if small feature refinements are added (e.g., tutorials, export/import of thresholds)

Bibliography

[1] N. Abdelhamid, F. Thabtah, and H. Abdeljaber.
Phishing detection: A recent intelligent machine learning comparison based on models content and features.
In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 72–77. IEEE, 2017.

[2] Akamai.
State of the internet / security report.
Technical report, 2024.

[3] R. Alabdan.
Phishing attacks survey: Types, vectors, and technical approaches.
Future Internet, 12(10):168, 2020.

[4] S. M. Albladi and G. R. S. Weir.
User characteristics that influence judgment of social engineering attacks in social networks.
Human-centric Computing and Information Sciences, 10(1):1–24, 2023.

[5] A. Aleroud and L. Zhou.
Phishing environments, techniques, and countermeasures: A survey.
Computers & Security, 68:81–100, 2023.

[6] A. H. Aljammal, S. Taamneh, A. Qawasmeh, and H. B. Salameh.
Machine learning based phishing attacks detection using multiple datasets.
International Journal of Interactive Mobile Technologies (iJIM), 17(5):71–88, 2023.

[7] A. Almomani et al.
Phishing websites detection using machine learning and deep learning techniques: A comprehensive analysis.

Electronics, 11(9):1413, 2023.

[8] M. Alsajri and H. AlKahtani.
Phishing detection using ml and dl: A systematic review.
IEEE Access, 12:42214–42232, 2024.

[9] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang.
A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities.
IEEE Communications Surveys & Tutorials, 21(2):1851–1877, 2020.

[10] Anti-Phishing Working Group.
Phishing activity trends report, q4 2024.
Technical report, 2025.

[11] AV-Comparatives.
Anti-phishing certification test.
Technical report, 2022.

[12] AV-Comparatives.
Anti-phishing protection test.
Technical report, 2024.

[13] Avast.
Avast online security & privacy technical documentation.
Technical report, 2024.

[14] P. A. Barraclough, G. Fehringer, and J. Woodward.
Intelligent cyber-phishing detection for online.
Computers & Security, 104:102123, 2021.

[15] Bitdefender.
Trafficlight technical whitepaper.
Technical report, 2024.

[16] British Psychological Society (BPS).
Code of human research ethics.
Technical report, 2014.

[17] A. Brown et al.
Emerging phishing vectors in augmented reality environments.
Journal of Cybersecurity, 9(1):243–261, 2023.

[18] Check Point Research.
Cyber security report.
In *Security Analysis*, 2024.

[19] J. Chen et al.
User experience evaluation of browser-based security extensions.
ACM Transactions on Privacy and Security, 24(3):78–96, 2021.

[20] T. Chen and R. Wang.
Api-based phishing: A new frontier in cybersecurity threats.
Journal of Cybersecurity, 10(2):110–128, 2024.

[21] K. L. Chiew et al.
Hybrid ensemble feature selection framework for phishing detection.
Information Sciences, 484:153–166, 2021.

[22] M. Cova, A. Prospero, and F. Zanoni.
Comparative analysis of anti-phishing browser extensions.
Computers & Security, 124:102950, 2023.

[23] DMARC.org.
Dmarc implementation report.
Technical report, 2024.

[24] FBI.
Internet crime report 2023.
Technical report, IC3, 2024.

[25] E. Fernandes, A. Nayak, R. Khandelwal, and K. Fawaz.
Experimental security analysis of sensitive data access by browser extensions.
In *Proc. ACM Web Conf. (WWW)*, 2024.

[26] R. Gokul and F. M. Philip.
Phishing detection.
YMER, 21(6):405–412, 2022.

[27] Google.
Enhanced safe browsing: Protection by default.
Google Security Blog, 2024.

[28] Google Transparency Report.
Safe browsing: Protecting web users for 15 years and counting, 2024.

[29] B. B. Gupta et al.
Fighting against phishing attacks: State of the art and future challenges.
Neural Computing and Applications, 2021.

[30] B. B. Gupta and R. Sharman.
Handbook of Computer Networks and Cyber Security: Principles and Paradigms.
Springer, 2022.

[31] J. Hong.
The state of phishing attacks.
Communications of the ACM, 65(5):40–42, 2022.

[32] IBM X-Force Threat Intelligence.
X-force threat intelligence index 2025.
Technical report, IBM Security, 2025.

[33] ISO.
Iso 9241-11:2018 ergonomics of human-system interaction — part 11: Usability: Definitions and concepts.
Technical report, International Organization for Standardization, 2018.

[34] A. K. Jain and B. B. Gupta.
Phishing detection: Analysis of visual similarity-based approaches.
Security and Communication Networks, 2017.

[35] A. K. Jain and B. B. Gupta.
Phishing detection: Analysis of visual similarity-based approaches.
Security and Privacy, 1(1):e9, 2018.

[36] A. K. Jain and B. B. Gupta.
A survey of phishing attack techniques.
Enterprise Information Systems, 15(9):1–39, 2021.

[37] D. Jampen et al.
A comparative literature review on anti-phishing training.
Human-centric Computing and Information Sciences, 2020.

[38] D. Jampen, G. Gür, T. Sutter, and B. Tellenbach.
Towards effective anti-phishing training.
Human-centric Computing and Information Sciences, 10(1):1–41, 2023.

[39] Kaspersky Labs.
Spam and phishing in 2022.
Technical report, 2023.

[40] R. Khandelwal, A. Nayak, and K. Fawaz.
Security landscape of browser extensions in 2024: Bypassing manifest v3 and stealing sensitive data.
In *Proc. ACM Web Conf. (WWW)*, 2024.

[41] M. Khonji, Y. Iraqi, and A. Jones.
Phishing detection: A literature survey.
IEEE Communications Surveys & Tutorials, 15(4):2091–2121, 2013.

[42] KnowBe4.
Phishing by industry benchmarking report.
Technical report, 2024.

[43] A. Kumar et al.
A novel ml-based phishing detection using url lexical features.
Expert Systems with Applications, 184:115464, 2022.

[44] A. Kumar and B. B. Gupta.
A survey on phishing detection and prevention techniques.
Journal of Network and Computer Applications, 192:103335, 2024.

[45] V. Kumar and D. Reddy.
Advanced phishing attacks: Trends and countermeasures.
IEEE Security & Privacy, 22(1):18–27, 2024.

[46] A. Kundu et al.
Deep learning approach for phishing detection.
Cybernetics and Systems, 51(3):292–312, 2020.

[47] LastPass Security.
The role of password managers in phishing prevention.
Technical report, 2023.

[48] Lookout.
Mobile phishing report.
Technical report, 2024.

[49] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury.
A survey of recent advances in deep learning models for detecting malware in desktop and mobile platforms.
ACM Computing Surveys, 56(6):Article 145, 2024.

[50] O. N. Mbadiwe, O. C. Nwokonkwo, A. I. Otuonye, C. O. Ikerionwu, and C. Etus.
Challenges of data collection and preprocessing for phishing email detection.
International Journal of Novel Research in Computer Science and Software Engineering, 11(2):45–59, 2024.

[51] McAfee.
Mobile threat report 2025.
Technical report, 2025.

[52] Meta.
Community standards enforcement report, q4 2024.
Technical report, 2025.

[53] Microsoft.
Digital defense report 2024.
Technical report, 2025.

[54] J. Misquitta and K. Anusha.
A comparative study of malicious url detection: Regular expression analysis, machine learning, and virustotal api.
2023.

[55] N. Nadia, W. Leewando, J. Paulus, and V. Nooril.
Phishing detection applications for website and domain at browser using virustototal api.
Engineering, Mathematics and Computer Science (EMACS) Journal, 5(2):93–96, 2023.

[56] A. Nayak, R. Khandelwal, E. Fernandes, and K. Fawaz.
Experimental security analysis of sensitive data access by browser extensions.
In *ACM WWW*, 2024.

[57] NCC Group.
Supply chain security: Trends and mitigations.
Technical report, 2024.

[58] Netcraft.
Anti-phishing extension technical documentation.
Technical report, 2024.

[59] T. V. Nguyen, H. N. Tran, and D. H. Le.
A hybrid phishing detection model using dom-based heuristics and text classification.
In *MADWeb Conference 2022*, 2022.

[60] J. Nielsen.
Usability Engineering.
Academic Press, Boston, MA, 1994.

[61] K. Nirmal, B. Janet, and R. Kumar.
A comprehensive study of phishing attacks.
Frontiers in Computer Science, 2:1–17, 2023.

[62] D. A. Norman.
The Design of Everyday Things: Revised and Expanded Edition.
Basic Books, New York, NY, 2013.

[63] L. Olejnik, C. Castelluccia, and A. Janc.
Why johnny can't browse in peace: On the uniqueness of web browsing history patterns.
Communications of the ACM, 58(2):50–57, 2015.

[64] OWASP.
Owasp top ten web application security risks.
Technical report, 2023.

[65] OWASP Foundation.
Phishing detection cheat sheet.
OWASP Cheat Sheet Series.

[66] OWASP Foundation.
Owasp top ten 2021, 2021.

[67] OWASP Foundation.
Content security policy cheat sheet, 2023.

[68] PhishDetect.
Technical documentation.
GitHub Repository, 2024.

[69] Proofpoint.
State of the phish report 2024.
Technical report, 2024.

[70] N. Rastogi and A. Sharma.
Evolution of phishing attacks: Empirical analysis.
International Journal of Information Security, 24:107–123, 2023.

[71] J. Reynolds and C. Rainear.
Understanding deceptive web design: Measuring the impact of domain-based mimicry on phishing susceptibility.
In *Proc. SoICT*, 2025.

[72] D. Sahoo, C. Liu, and S. C. H. Hoi.
Malicious url detection using machine learning: A survey.
arXiv preprint arXiv:1701.07179, 2017.

[73] SANS Institute.
Security awareness report.
Technical report, 2024.

[74] H. Sharma et al.
Comparative analysis of phishing detection tools.
In *2020 International Conference on Confluence*, 2020.

[75] R. Singh et al.
Secure chrome extension development with manifest v3: A practical review.
Journal of Web Security, 8(1):12–27, 2025.

[76] R. Singh et al.
Secure chrome extension development with manifest v3: A practical review.
Journal of Web Security, 8(1), 12–27, 2025.

[77] S. Singh, G. Varshney, and V. Mishra.
A study on malicious browser extensions in 2025.
arXiv preprint arXiv:2503.04292, 2025.

[78] SoICT.
Improving phishing detection using in-browser structural analysis.
In *Proc. 14th Int. Symp. Inf. Commun. Technol.*, 2023.

[79] SSL Labs.
Ssl pulse: Survey of ssl implementation.
Technical report, 2024.

[80] StatCounter.
Desktop browser market share worldwide, 2024.

[81] Symantec.
Internet security threat report, vol. 24.
Technical report, 2023.

[82] A. A. Ubing et al.
Evaluation of phishing detection using decision tree and svm.
Journal of Computer Networks and Communications, 2021.

[83] G. Varshney and J. M. Chatterjee.
Assessment of anti-phishing browser extensions.
ACM Computing Surveys, 55(1):1–38, 2023.

[84] G. Varshney et al.
Comprehensive assessment of browser extensions for phishing prevention.
ACM Computing Surveys, 55(1), 2022.

[85] Verizon.
Data breach investigations report.
Technical report, 2023.

[86] Verizon.
Data breach investigations report.
Technical report, 2024.

[87] M. Vijayalakshmi, S. M. Shalinie, M. H. Yang, and R. M. U.
Web phishing detection techniques: A survey on the state-of-the-art, taxonomy and future directions.
IET Networks, 9(5):235–246, 2020.

[88] VirusTotal.
Virustotal public api v2.0, 2024.

[89] D. Wang and X. Liu.
Visualphish: Phishing detection using visual similarity.
IEEE Trans. on Dependable and Secure Computing, 20(1):187–201, 2023.

[90] H. Wang.
Automated phishing detection using urls and webpages.
arXiv preprint arXiv:2408.01667, 2024.

[91] Y. Wang.
Practical implications of tls certificate policies on secure communications.
IEEE Transactions on Information Forensics and Security, 19:1123–1134, 2024.

[92] Y. Wang.
Practical implications of tls certificate policies on secure communications.
IEEE Transactions on Information Forensics and Security, 19, 1123–1134, 2024.

[93] R. Wash and M. M. Cooper.
Who provides phishing training?
In *Proceedings of the CHI Conference*, 2021.

[94] E. J. Williams and D. Polage.
Persuasiveness of phishing email.
Behaviour & Information Technology, 38(2):184–197, 2023.

[95] Q. Yaseen, Y. Jararweh, and M. Al-Ayyoub.
Phishing in ar: Threats and countermeasures.
IEEE Access, 12:78210–78225, 2024.

[96] O. Yavanoglu and M. Aydos.
A review on cyber security datasets for machine learning algorithms.
In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2186–2193. IEEE, 2017.

[97] O. Yavanoglu and M. Aydos.
A review on cyber security datasets.
In *Proc. IEEE Big Data*, pages 2186–2193, 2022.

[98] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K. K. R. Choo.
Phishing detection and threat mitigation using machine learning techniques.
IEEE Access, 10:123456–123468, 2022.

[99] J. Zhou et al.
Integrating threat intelligence apis for real-time phishing detection.
Cybersecurity Research Review, 7(3):45–58, 2023.